Zürcher Hochschule
für Angewandte Wissenschaften

School of
Engineering

InIT Institut für angewandte
Informationstechnologie

## Bachelor's Thesis Computer Science

# Document Digitization for Chess Scorecards

| | |
|---|---|
| **Authors** | Béla Horváth<br>Colin Dreher |
| **Supervisors** | Prof. Dr. Mark Cieliebak |
| **Date** | 19.06.2020 |

Zürcher Hochschule
für Angewandte Wissenschaften

School of
Engineering

# Erklärung betreffend das selbstständige Verfassen einer Bachelorarbeit an der School of Engineering

## Erklärung betreffend das selbstständige Verfassen einer Bachelorarbeit an der School of Engineering

Mit der Abgabe dieser Bachelorarbeit versichert der/die Studierende, dass er/sie die Arbeit selbständig und ohne fremde Hilfe verfasst hat. (Bei Gruppenarbeiten gelten die Leistungen der übrigen Gruppenmitglieder nicht als fremde Hilfe.)
Der/die unterzeichnende Studierende erklärt, dass alle zitierten Quellen (auch Internetseiten) im Text oder Anhang korrekt nachgewiesen sind, d.h. dass die Bachelorarbeit keine Plagiate enthält, also keine Teile, die teilweise oder vollständig aus einem fremden Text oder einer fremden Arbeit unter Vorgabe der eigenen Urheberschaft bzw. ohne Quellenangabe übernommen worden sind.

Bei Verfehlungen aller Art treten die Paragraphen 39 und 40 (Unredlichkeit und Verfahren bei Unredlichkeit) der ZHAW Prüfungsordnung sowie die Bestimmungen der Disziplinarmassnahmen der Hochschulordnung in Kraft.

| **Ort, Datum:** | **Name Studierende:** |
|---|---|
| Winterthur, 19.06.2020 | Béla Horváth |
| Winterthur, 19.06.2020 | Colin Dreher |

# Zusammenfassung

Schachpartien werden von Schachspielern an Turnieren oder im Training auf sogenannten “Scorecards” in einer genormten Syntax notiert. Diese Scorecards variieren pro Anlass in ihrem Aufbau. Damit diese zu einem späteren Zeitpunkt mit einem Computerprogramm analysiert werden können, müssen die Schachzüge von Hand in eine Analysesoftware übertragen werden, was mühsam und zeitaufwendig ist. Momentan gibt es auf dem Markt keine Lösung, welche diesen Digitalisierungsprozess übernimmt, ohne ein genormtes Scorecard Format zu verwenden. In dieser Arbeit wurde ein System entwickelt, welches dem Benutzer diesen Digitalisierungsprozesses abnimmt. Es wurde eine Webanwendung entwickelt, in welcher ein Bild hochgeladen wird und eine “Portable Game Notation” (PGN) Datei zurückerhalten wird. Die bei der Digitalisierung erkannten Schachzüge werden dem Benutzer zur Kontrolle präsentiert und der Benutzer kann im Nachhinein auf die Erkennung Einfluss nehmen, um allfällige Fehler zu beheben. Die Arbeit wurde in Problemstellungen aufgeteilt und iterativ erarbeitet, mit dem Ziel, eine komplette Pipeline zu erzeugen. Danach wurden die einzelnen Schritte im Algorithmus an den Stellen optimiert, wo die grössten Verbesserungen im Prozess erzielt werden konnten. Die erzeugte “Proof of Concept” Applikation verwendet für die Handschriftenerkennung die ABBYY Cloud OCR SDK API (ABBYY). Zusätzlich wurde ein zweiter Ansatz implementiert, in welchem in weiteren Schritten eigene, neuronale Netze als Klassifizierer trainiert und eingesetzt werden können. Die implementierten Teile wurden mittels einer selbst erstellten Datenserie evaluiert. Das Resultat zeigte, dass die von ABBYY erhaltenen Erkennungen für schön geschriebene Scorecards bereits gute Übereinstimmungen liefern. Diese konnten mit der Hilfe von Entscheidungsbäumen und Heuristiken weiter verbessert werden. Gesamt entscheidet jedoch die Schrifterkennung, ob die Digitalisierung erfolgreich oder fehlerbehaftet ist. Die Applikation bietet eine solide Basis, auf welcher aufgebaut werden muss, um ein marktfähiges Produkt zu erzeugen.

## Abstract

Chess games are noted by chess players at tournaments or in training on so-called "scorecards" in a standardized syntax. These scorecards vary in their structure for each occasion. To later analyse them with a computer program, the chess moves must be transferred manually to analysis software, which is tedious and time-consuming. Currently there is no solution on the market that can take over this digitization process without using a standardized scorecard format. In this thesis a system was developed, which relieves the user of this digitization process. A web application was developed in which an image is uploaded and a "Portable Game Notation" (PGN) file is retrieved. The chess moves detected during the digitization process are presented to the user for control and the user can influence the detection afterwards to correct any errors. The work was divided into problems and worked out iteratively, with the aim of creating a complete pipeline. Then the individual steps in the algorithm were optimized at those points where the greatest improvements in the process could be achieved. The generated "Proof of Concept" application uses the ABBYY Cloud OCR SDK API (ABBYY) for handwriting recognition. In addition, a second approach was implemented, in which own neural networks can be trained and used as classifiers in further steps. The implemented parts of the algorithm were evaluated using a self-created data set. The result showed that the recognitions received from ABBYY for beautifully written scorecards already provide good matches. These could be further improved with the help of decision trees and heuristics. Overall, however, the character recognition decides whether the digitization is successful or not. The application offers a solid basis on which to build to create a marketable product.

# Preface

We are Colin Dreher and Béla Horváth. During our last semester as bachelor students we could apply and consolidate the knowledge we had learned during our studies in the fields of software development, image processing and artificial intelligence in this thesis. It was a challenge but offered more opportunities to build on our own ideas.

We would like to thank Prof. Dr. Mark Cieliebak very much. Thanks to the good supervision and the great discussions, exciting topics and interesting suggestions could be discussed every week. Despite the Corona crisis, the collaboration was continued and we always found an open ear when questions arose. We would also like to thank Anand for sharing his ideas with us. Through him a direct connection to a customer of such a software could be established and this bachelor thesis would not have been possible without him. And finally, we would like to thank all the people who have voluntarily agreed to fill in scorecards for the evaluation.

Winterthur, June 19, 2020

## Table of contents

# 1 Introduction

## 1.1 Initial situation

Nowadays chess players write down the moves of a chess game in a Standard Algebraic Notation (SAN) on a scorecard while they are playing. If the players want to analyse their game, they must replay all the moves by hand in a tool as for instance "Chess.com". This takes a lot of time but can give the players valuable information about their behavior and tactics. Currently the existing standard for scorecard scanning is called "Reine – Chess"[1]. To use the software, a player must use their standardized scorecard template to automatically read and digitize all the moves. This limits individual players because tournaments often have their own scorecard format which must be used at their event. Currently the only product on the market that follows the approach to read in different scorecards with tables is the app "CheSScan"[2]. It allows to take a picture of a scorecard which is then processed and output as "Portable Game Notation" (PGN). However, the product is not yet fully developed, since the user is not allowed to touch the table lines with his handwriting and must write legibly in block letters with sufficient distance. The reason for this is that such conditions make image recognition very difficult and current state of the art approaches still struggle to make handwritten character recognition in tables work. Furthermore, it might be objected why one does not write the moves directly on an electronic device. However, at official tournaments it is forbidden to use any electronic devices, unless it is considered a "chess notation device"[3]. These devices must be officially certified which limits the usage and market of potential buyers since it is an extra device and not only a service.

## 1.2 Goal and task

The aim of this bachelor thesis is to implement an algorithm which processes an image of a scorecard. It should detect the handwritten moves, recognize them with optical/intelligent character recognition (OCR/ICR), check them for correctness and return them as a machine readable PGN.

[1] Reine - Scannable chess scoresheets. Available: https://www.reinechess.com/ [20.03.2020]
[2] CheSScan - Chess scoresheet scanner. Available: https://chesscan.com/ [28.03.2020]
[3] ChessNoteR – The future of chess notation. Available: https://www.chessnoter.com/ [28.05.2020]

The algorithm should be implemented in Python on a web server and be usable in a web application with an interface for the user to interact with. The possibility of correcting unclear or incorrect moves is also provided to the user when interacting with the PGN output.

## 1.3 Overview of the thesis

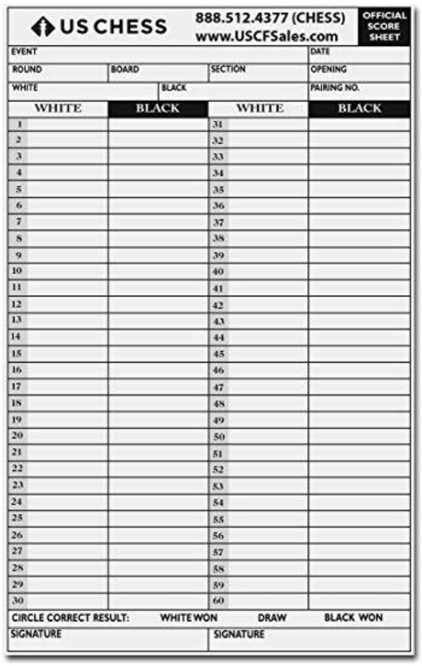
Initially, the current competition and the software available on the market are discussed. Afterwards, the final product is presented within the application showcase. After the showcase, the focus lies on the encountered and solved challenges inside of the implementation chapter. This is the main part of the thesis. Each subchapter contains the necessary theory, methodology, evaluation, results and a discussion. Further information about the software architecture and the deployment of the application are in the appendix in chapter 9.2 and are not a central part of the work.

## 1.4 Reduction of the scope

This thesis focuses on creating an algorithm and a proof of concept implementation. To reduce the scope of the thesis, certain steps were simplified and need further follow up work to produce a marketable product. These simplifications include multiple users working simultaneously and the support of scorecards without a table layout. Additionally, while different chess notations exist, the request was to support the PGN syntax [1]. PGN is a data format which is used to store and share chess games in an ASCII text file. The PGN syntax is mostly based on SAN [2] which uses single character abbreviations for chess pieces like “K” for king and “Q” for queen etc.

Due to the reduction of the scope, the algorithm is only developed for scorecards with solid table lines in a similar form as seen in Figure 1 or Figure 2. Layouts, as in Figure 3 are not supported.

Tables can also be separated by whitespace but they must be visible and fully connected at the top and bottom of the moves. This layout was chosen after receiving confirmation from the Swiss Chess Federation that it was the most common.



US CHESS 888.512.4377 (CHESS) www.USCFSales.com OFFICIAL SCORE SHEET

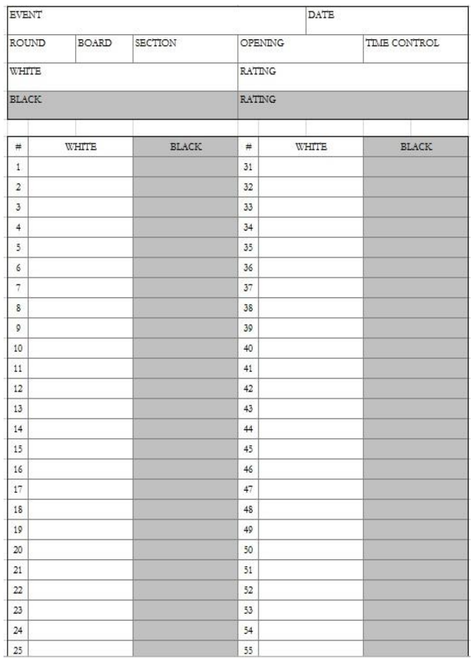
| | | | |
|---|---|---|---|
| EVENT | | | DATE |
| ROUND | BOARD | SECTION | OPENING |
| WHITE | BLACK | | PAIRING NO. |

| | WHITE | BLACK | | WHITE | BLACK |
|---|---|---|---|---|---|
| 1 | | | 31 | | |
| 2 | | | 32 | | |
| 3 | | | 33 | | |
| 4 | | | 34 | | |
| 5 | | | 35 | | |
| 6 | | | 36 | | |
| 7 | | | 37 | | |
| 8 | | | 38 | | |
| 9 | | | 39 | | |
| 10 | | | 40 | | |
| 11 | | | 41 | | |
| 12 | | | 42 | | |
| 13 | | | 43 | | |
| 14 | | | 44 | | |
| 15 | | | 45 | | |
| 16 | | | 46 | | |
| 17 | | | 47 | | |
| 18 | | | 48 | | |
| 19 | | | 49 | | |
| 20 | | | 50 | | |
| 21 | | | 51 | | |
| 22 | | | 52 | | |
| 23 | | | 53 | | |
| 24 | | | 54 | | |
| 25 | | | 55 | | |
| 26 | | | 56 | | |
| 27 | | | 57 | | |
| 28 | | | 58 | | |
| 29 | | | 59 | | |
| 30 | | | 60 | | |

CIRCLE CORRECT RESULT: WHITE WON DRAW BLACK WON

SIGNATURE SIGNATURE

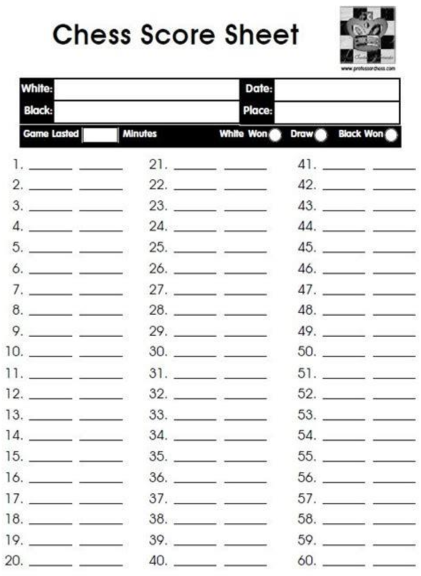
Figure 1 USCF scorecard layout with a closed table.[4]



| | | | | |
|---|---|---|---|---|
| EVENT | | | DATE | |
| ROUND | BOARD | SECTION | OPENING | TIME CONTROL |
| WHITE | | | RATING | |
| BLACK | | | RATING | |

| # | WHITE | BLACK | # | WHITE | BLACK |
|---|---|---|---|---|---|
| 1 | | | 31 | | |
| 2 | | | 32 | | |
| 3 | | | 33 | | |
| 4 | | | 34 | | |
| 5 | | | 35 | | |
| 6 | | | 36 | | |
| 7 | | | 37 | | |
| 8 | | | 38 | | |
| 9 | | | 39 | | |
| 10 | | | 40 | | |
| 11 | | | 41 | | |
| 12 | | | 42 | | |
| 13 | | | 43 | | |
| 14 | | | 44 | | |
| 15 | | | 45 | | |
| 16 | | | 46 | | |
| 17 | | | 47 | | |
| 18 | | | 48 | | |
| 19 | | | 49 | | |
| 20 | | | 50 | | |
| 21 | | | 51 | | |
| 22 | | | 52 | | |
| 23 | | | 53 | | |
| 24 | | | 54 | | |
| 25 | | | 55 | | |

Figure 2 A different scorecard layout with a closed table.[5]



Chess Score Sheet

| | | | |
|---|---|---|---|
| White: | | Date: | |
| Black: | | Place: | |

Game Lasted ____ Minutes White Won ○ Draw ○ Black Won ○

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 1. | | | 21. | | | 41. | | |
| 2. | | | 22. | | | 42. | | |
| 3. | | | 23. | | | 43. | | |
| 4. | | | 24. | | | 44. | | |
| 5. | | | 25. | | | 45. | | |
| 6. | | | 26. | | | 46. | | |
| 7. | | | 27. | | | 47. | | |
| 8. | | | 28. | | | 48. | | |
| 9. | | | 29. | | | 49. | | |
| 10. | | | 30. | | | 50. | | |
| 11. | | | 31. | | | 51. | | |
| 12. | | | 32. | | | 52. | | |
| 13. | | | 33. | | | 53. | | |
| 14. | | | 34. | | | 54. | | |
| 15. | | | 35. | | | 55. | | |
| 16. | | | 36. | | | 56. | | |
| 17. | | | 37. | | | 57. | | |
| 18. | | | 38. | | | 58. | | |
| 19. | | | 39. | | | 59. | | |
| 20. | | | 40. | | | 60. | | |

Figure 3 A scorecard layout with an open table layout.[6]

[4] Official US chess Self-Duplicating Scorecards. Available: https://www.uscfsales.com/carbonless-score-sheets-single-sheet.html, [30.03.2020]
[5] 13 Free Sample Chess Scorecards. Available: https://www.printablesample.com/wp-content/uploads/2017/01/chessnotation-1.jpg [30.03.2020]
[6] 13 Free Sample Chess Scorecards. Available: https://www.printablesample.com/wp-content/uploads/2017/01/chess-score-sheet-13.jpg [30.03.2020]

# 2 State of the art

This chapter analyses which solutions already exist on the market. It also discusses the direction in which the development of the application will be steered and which features are prioritized.

## 2.1 Competitor analysis

In the following, the existing scorecard digitization services and solutions for end users are investigated in a competitor analysis. The analysis of the individual providers focuses on the services offered and the restrictions. The prices, the supported scorecard formats, and other features are listed.

### 2.1.1 Reine – Chess

"Reine – Chess" is a free web service. It offers the possibility to download a scorecard in the format seen in Figure 4 and to upload it when filled out. Due to their own format the sources of error are reduced. On each corner of the image, an "ArUco" mark [3] can be spotted. These marks allow for a transformation of the image since those marks can always be located within an image with their coordinates and rotation. This leads to perfectly aligned images, which is needed for the chess move recognition if the table wants to be used as a guideline. The rough procedure of the application is as follows:

1. Take image
2. Align image with "ArUco" markers
3. Cut the individual characters
4. Pre-process characters for convolutional neural network (CNN)
5. Classification by CNN (custom CNN per chess move length)
6. Post-processing the classification
7. Download PGN

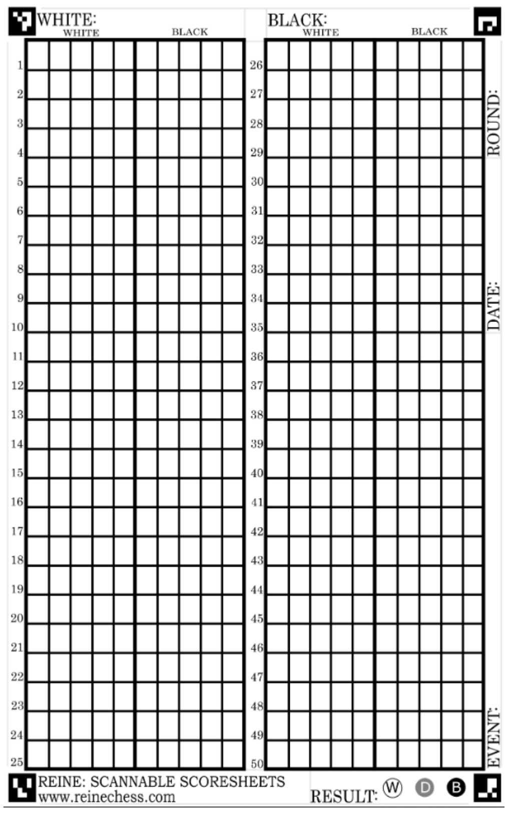


Figure 4 The downloadable scorecard format from Reine – Chess [7] with ArUco markers in the corners.

When visiting the source code[8], each character is analysed individually, and this is implemented using different convolutional neural networks (CNN) trained with different training data based on the EMNIST data set [4]. This improves the accuracy of the recognition since it is known what patterns are possible in each two to five-character long chess move. No extra validation regarding the board state is done in the current implementation.

**Limitations:**

- Currently only supports their own scorecard format
- Move can only have five characters and no special characters i.e. "#, =, +, -"
- Currently no mobile application exists

[7] Scannable Scoresheets. Available: https://77614886-b378-4a7e-8da3-e37d91040160.filesusr.com/ugd/384def_8971d714115343ccb8db211816f057eb.pdf [05.04.2020]
[8] GitHub repository of Reine – Chess. Available: https://github.com/Messier-16/Reine-Chess-Scoresheet-Scanner [20.03.2020]

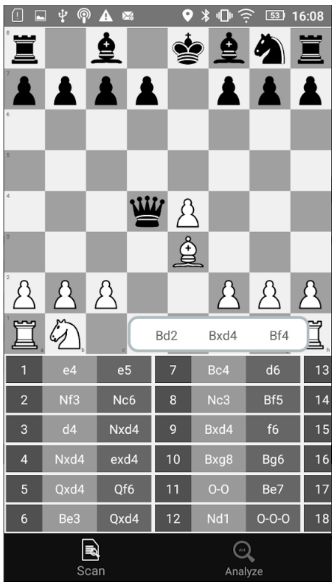
### 2.1.2 CheSScan

"CheSScan" is a mobile application available for iOS and android. The first step while using the app is to take a picture of a scorecard. Secondly the image is automatically analysed and the detected moves are displayed as shown in Figure 5. Afterwards the user can correct them individually by overwriting the moves or by playing a move with a chess piece. Finally, the PGN can be output or the game can be analysed.

Since "CheSScan" is not an open source project, there is no freely available code. Through a conversation with the developer it turned out that he follows similar approaches to those used in the application "Reine – Chess".

**Limitations:**

- It is only available as a mobile application in the app and play store
- The user cannot see why the app does not recognize anything or only a part
- Unable to recognize cursive written text
- Not fully developed, no manual correction feature



| | | | | | | |
|---|---|---|---|---|---|---|
| 1 | e4 | e5 | 7 | Bc4 | d6 | 13 |
| 2 | Nf3 | Nc6 | 8 | Nc3 | Bf5 | 14 |
| 3 | d4 | Nxd4 | 9 | Bxd4 | f6 | 15 |
| 4 | Nxd4 | exd4 | 10 | Bxg8 | Bg6 | 16 |
| 5 | Qxd4 | Qf6 | 11 | O-O | Be7 | 17 |
| 6 | Be3 | Qxd4 | 12 | Nd1 | O-O-O | 18 |

Figure 5 Screenshot of CheSScan Android App interface[9].

[9] CheSScan Android App. Available: https://play.google.com/store/apps/details?id=com.nagdev.alok.chesscan [05.04.2020]

### 2.1.3 Results

The competitor analysis shows that only the reviewed two applications exist. "Reine – Chess" offers a working solution that is under development which only works for their scorecard format. In practice, tournament organizers use their own scorecards or standardized templates of the local chess federation. On the other hand, "CheSScan" offers a general solution that supports various scorecard layouts but does not provide reliable detection. While testing it was unclear when or where an error occurred, and it sometimes failed after less than five moves. Furthermore, neither of the applications, present the ability to trace the source of the failed recognition back to the responsible step. This is important as it enhances the user experience and helps to avoid common mistakes in later uploads.

To present a solution that takes as many steps from the user as possible and informs him if something fails, this is also considered in the implementation. Whenever an error occurs, its source must be clearly communicated to the user.

# 3 Methods and procedures

## 3.1 Workflow

The software produced in this thesis is called "Very Chess" and was developed using a semi-agile approach. Meetings with the supervisor were held in intervals of one to two weeks in which the progress was discussed. A trelloboard[10] was used to maintain the overview of open tasks and priorities. To ensure code versioning and collaboration, Git was used in combination with GitHub[11]. At the beginning of the thesis, questions regarding the task, goal and scope were clarified with the thesis supervisor. To get an idea of what is possible, a meeting with an external expert was held. The collected information from the meeting was compiled and discussed with the supervisor. Since the task was only roughly predefined, the scope had to be limited.

The aim during implementation was to have a complete pipeline as quickly as possible. The steps should at least provide a correct and robust output for simple table designs and well legible handwriting. Comparisons with the state of the art are made and alternative approaches are pursued. Everything that can already be solved with the help of libraries is implemented.

[10] Trello. Available: https://trello.com/ [17.06.2020]
[11] GitHub. Available: https://github.com/ [17.06.2020]

## 3.2 Approach

Based on the competitor analysis shown in chapter 2.1, "Reine – Chess" implements the most promising approach. In this thesis a similar approach is followed. However, since "Reine – Chess" uses a special scorecard, some steps must be implemented differently.

- General scorecards do not have "ArUco" markers, a different alignment approach must be chosen.
- The scorecards have unknown layouts. Therefore, an algorithm must be implemented that extracts the written moves inside of the tables.
- The letters in scorecards are not individually located in fields. Thus, they must be recognized and separated from each other or read as a whole "word".
- Whole words could be recognized with an ICR/OCR engine instead of single characters.
- The post-processing approach of "Reine – Chess" relies on the length of the recognition. This is robust since all characters are individually localized and the length of the move is therefore correct. This is not possible in general table layouts.

Considering the alternative implementations stated above, **the following approach is defined:**

1. Image alignment (align the uploaded image before continuing with the pre-processing)
2. Table extraction (filter noise in the scorecard and extract the table)
3. Chess move localisation (locate the relevant boxes that contain chess moves)
4. Recognize chess moves with ICR
    a. CNN approach (pre-process and implement a custom CNN)
    b. ICR engine implementation (select and implement a fitting ICR engine/API)
5. Improvement of recognition (further improve the recognition of the implemented ICR engine)
6. Present PGN and download PGN (web interface to interact with the output and to download the output)

Furthermore, intermediate results of the algorithm (the steps above) are presented to the user for validation. The presented results can be confirmed, corrected or rejected to guarantee a successful recognition and prediction. The steps 1-5 can be mapped to the steps of the Implementation chapter and will be discussed in detail in their respective subchapters.

# 4 Application showcase

To ensure the overview over the workflow, the final application is presented first. The whole process is described from the user's perspective. The workflow starts with uploading an image to the web server and ends after the user decides to download the resulting PGN file (Figure 6). The process is presented in a simplified form and is clearly defined in the following chapters. Further information about the chosen software architecture is in the appendix in chapter 9.2.

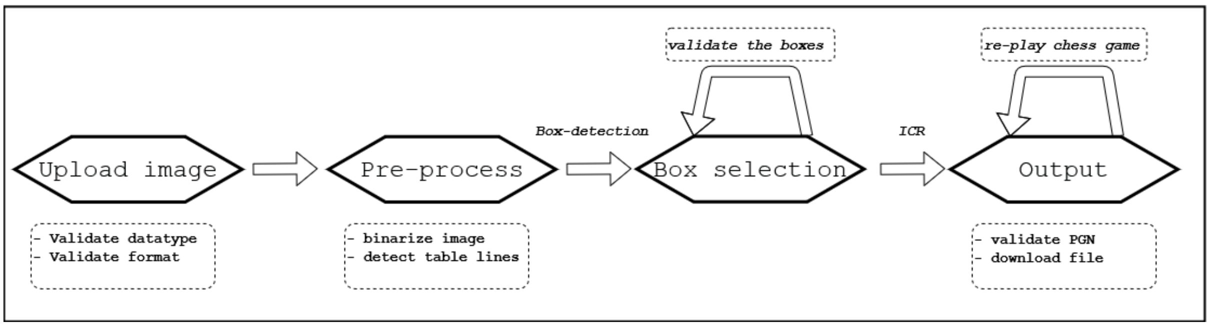
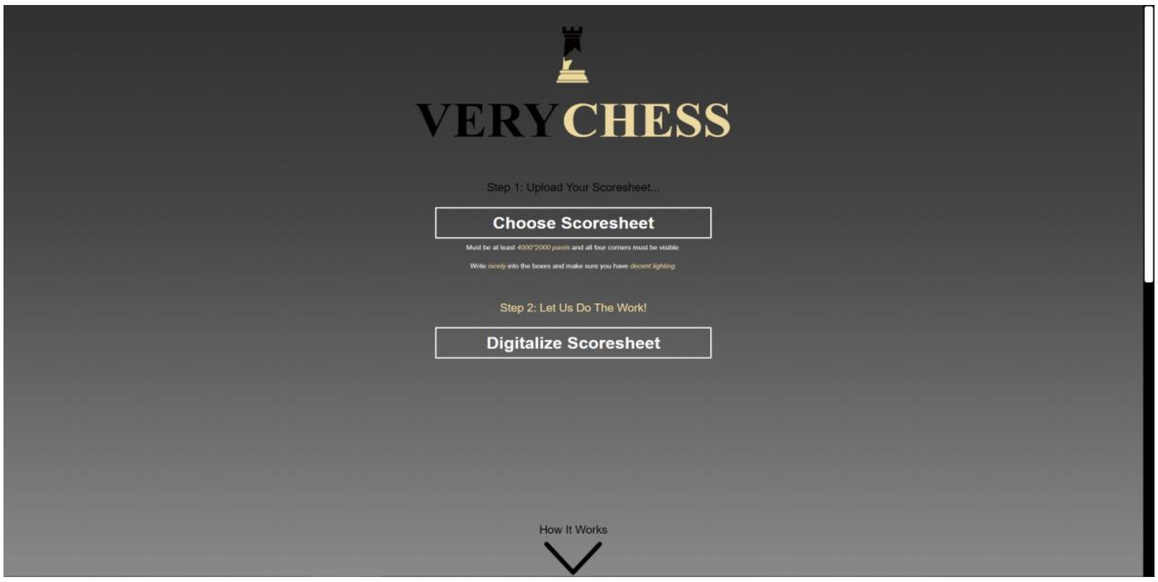


Figure 6 The general workflow of the application. Each main step is in a hexagon.

## 4.1 Upload image

First, with a press of the “Choose Scorecard” button an image of a scorecard must be chosen where all four corners of the sheet are visible. The image is uploaded with a press of “Digitalize Scorecard”. Only “.jpeg” and “.png” file formats can be uploaded. It is possible to directly take an image with a mobile phone. The resolution of the selected image must be above 2000x1500 pixel.


VERYCHESS
Step 1: Upload Your Scoresheet...
Choose Scoresheet
Step 2: Let Us Do The Work!
Digitalize Scoresheet
How It Works

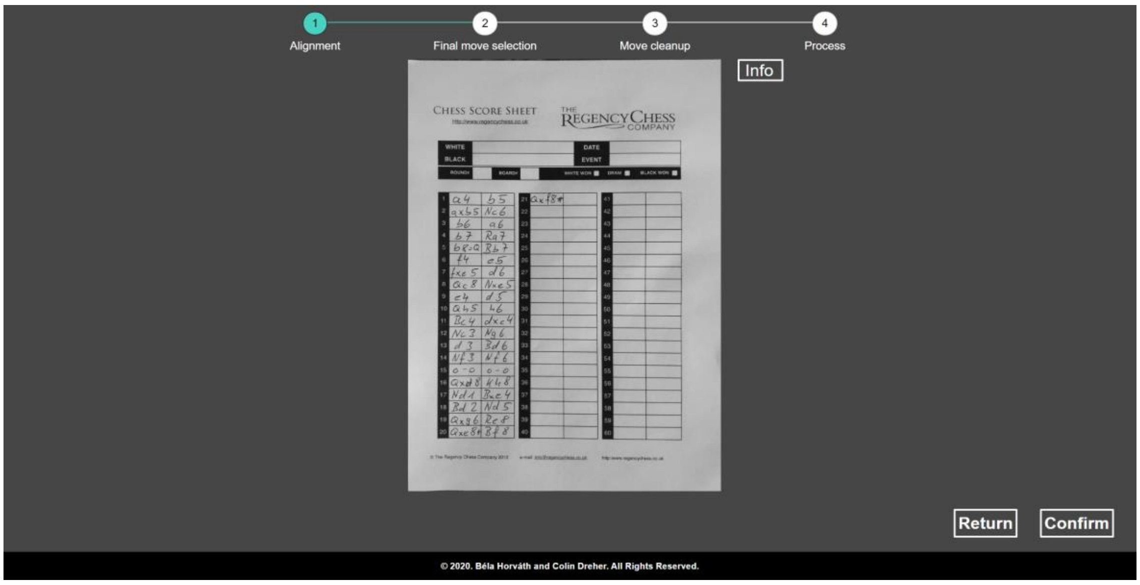
Must be at least *4000*2000 pixels* and all four corners must be visible

Write *nicely* into the boxes and make sure you have *decent lighting*

Figure 7 Landing page of the application where the user can upload an image and start the application.

## 4.2 Box selection

In Figure 8, the user is asked to validate that the image alignment worked. This step can be continued when the “Confirm” button is clicked. If the image is aligned incorrectly, the user must take a new image with better quality.



CHESS SCORE SHEET
http://www.regencychess.co.uk

THE REGENCY CHESS COMPANY

| | | | |
|---|---|---|---|
| WHITE | | DATE | |
| BLACK | | EVENT | |

ROUND ☐ BOARD ☐ WHITE WON ☐ DRAW ☐ BLACK WON ☐

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 1 | a4 | b5 | 21 | Qxf8# | | 41 | | |
| 2 | axb5 | Nc6 | 22 | | | 42 | | |
| 3 | b6 | a6 | 23 | | | 43 | | |
| 4 | b7 | Ra7 | 24 | | | 44 | | |
| 5 | b8=Q | Rb7 | 25 | | | 45 | | |
| 6 | f4 | e5 | 26 | | | 46 | | |
| 7 | fxe5 | d6 | 27 | | | 47 | | |
| 8 | Qc8 | Nxe5 | 28 | | | 48 | | |
| 9 | e4 | d5 | 29 | | | 49 | | |
| 10 | Qh5 | h6 | 30 | | | 50 | | |
| 11 | Bc4 | dxc4 | 31 | | | 51 | | |
| 12 | Nc3 | Ng6 | 32 | | | 52 | | |
| 13 | d3 | Bd6 | 33 | | | 53 | | |
| 14 | Nf3 | Nf6 | 34 | | | 54 | | |
| 15 | 0-0 | 0-0 | 35 | | | 55 | | |
| 16 | Qxd8 | Kh8 | 36 | | | 56 | | |
| 17 | Nd1 | Bxe4 | 37 | | | 57 | | |
| 18 | Bd2 | Nd5 | 38 | | | 58 | | |
| 19 | Qxg6 | Re8 | 39 | | | 59 | | |
| 20 | Qxe8# | Bf8 | 40 | | | 60 | | |

© The Regency Chess Company 2013 e-mail info@regencychess.co.uk http://www.regencychess.co.uk

© 2020. Béla Horváth and Colin Dreher. All Rights Reserved.

Figure 8 The aligned image is displayed in the page. The user can confirm the correctness of the alignment.

After the first confirmation, the user is asked to select his “final move”. This corresponds to the last handwritten chess move on his scorecard. After the selection, the colours will indicate which parts are still active as shown on the right side of Figure 9.

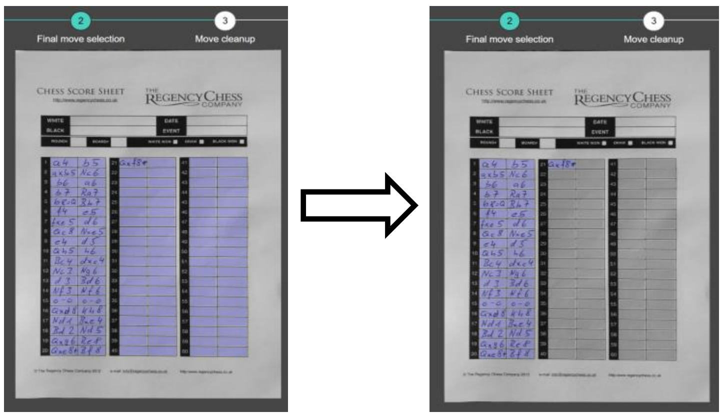


Figure 9 The user selects the last move and automatically the rest of the boxes are unselected.

With a click on “Confirm” the user is then asked to unselect any boxes that do not belong to his chess moves. This is only the necessary in certain scorecard layouts and thus not needed if everything looks correct as it is shown in Figure 10. Figure 11 on the other hand shows the deselection process of the boxes containing “White” and “Black” which do not belong to the chess moves.

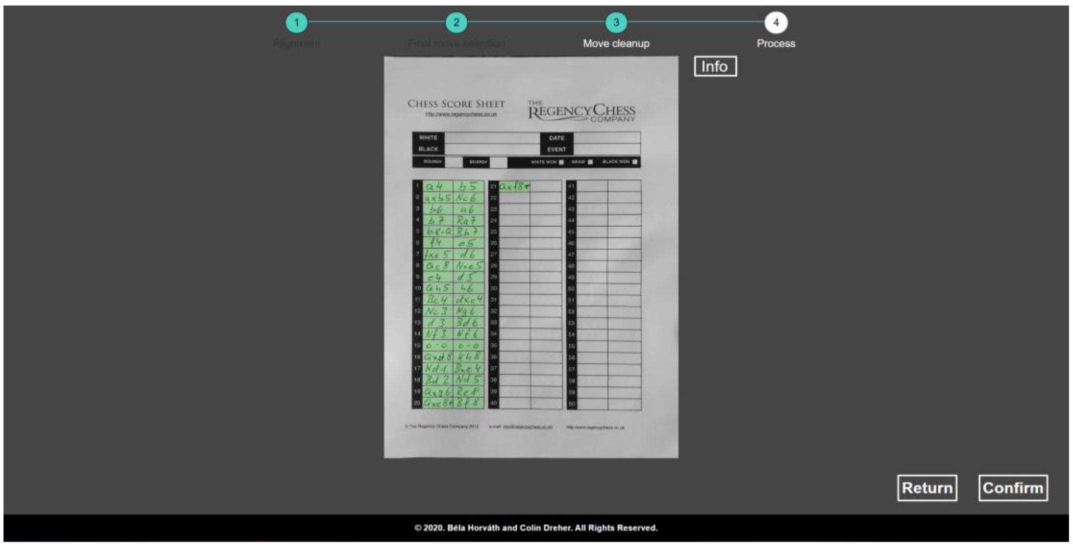
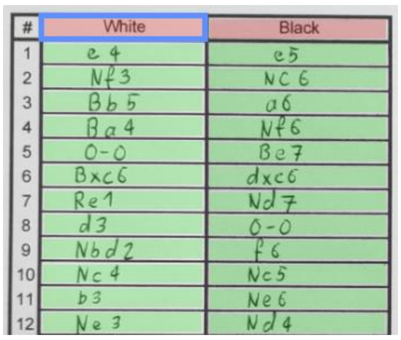

Alignment
Final move selection
Move cleanup
Process
Info
Chess Score Sheet
The Regency Chess Company
Return
Confirm
© 2020. Béla Horváth and Colin Dreher. All Rights Reserved.

Figure 10 The user selects any unwanted boxes. This example does not contain any unwanted boxes.

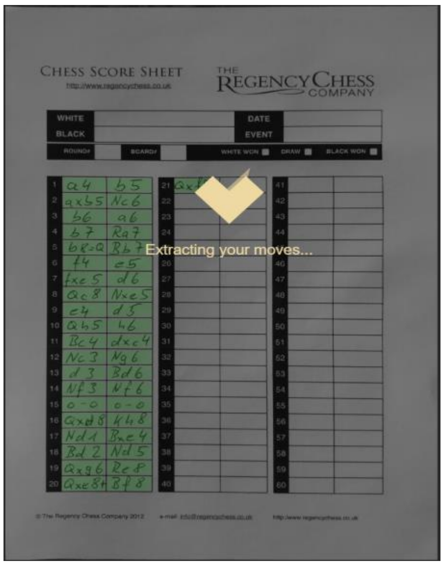


| # | White | Black |
|---|---|---|
| 1 | e4 | e5 |
| 2 | Nf3 | NC6 |
| 3 | Bb5 | a6 |
| 4 | Ba4 | Nf6 |
| 5 | 0-0 | Be7 |
| 6 | Bxc6 | dxc6 |
| 7 | Re1 | Nd7 |
| 8 | d3 | 0-0 |
| 9 | Nbd2 | f6 |
| 10 | Nc4 | Nc5 |
| 11 | b3 | Ne6 |
| 12 | Ne3 | Nd4 |

Figure 11 The unwanted boxes are deselected by the user and marked in red.

After validating that no wrongly highlighted boxes exist, the “Confirm” button can be pressed once more to send the selected boxes to the server. There they are recognized by ABBYY and based on the recognition a prediction is made for each move.

For every asynchronous step, a loading screen is displayed to inform the user that the application is working, seen in Figure 12.



CHESS SCORE SHEET
http://www.regencychess.co.uk

THE REGENCY CHESS COMPANY

| | | | |
|---|---|---|---|
| WHITE | | DATE | |
| BLACK | | EVENT | |
| ROUND# | | BOARD# | |
| WHITE WON | | DRAW | BLACK WON |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 1 | a4 | b5 | 21 | Qx | | 41 | | |
| 2 | axb5 | Nc6 | 22 | | | 42 | | |
| 3 | b6 | a6 | 23 | | | 43 | | |
| 4 | b7 | Ra7 | 24 | | | 44 | | |
| 5 | b8=Q | Rb7 | 25 | | | 45 | | |
| 6 | f4 | e5 | 26 | | | 46 | | |
| 7 | fxe5 | d6 | 27 | | | 47 | | |
| 8 | Qc8 | Nxe5 | 28 | | | 48 | | |
| 9 | e4 | d5 | 29 | | | 49 | | |
| 10 | Qb5 | h6 | 30 | | | 50 | | |
| 11 | Bc4 | dxc4 | 31 | | | 51 | | |
| 12 | Nc3 | Ng6 | 32 | | | 52 | | |
| 13 | d3 | Bd6 | 33 | | | 53 | | |
| 14 | Nf3 | Nf6 | 34 | | | 54 | | |
| 15 | o-o | o-o | 35 | | | 55 | | |
| 16 | Qxd8 | Kh8 | 36 | | | 56 | | |
| 17 | Nd1 | Bxe4 | 37 | | | 57 | | |
| 18 | Bd2 | Nd5 | 38 | | | 58 | | |
| 19 | Qxg6 | Re8 | 39 | | | 59 | | |
| 20 | Qxe8+ | Bf8 | 40 | | | 60 | | |

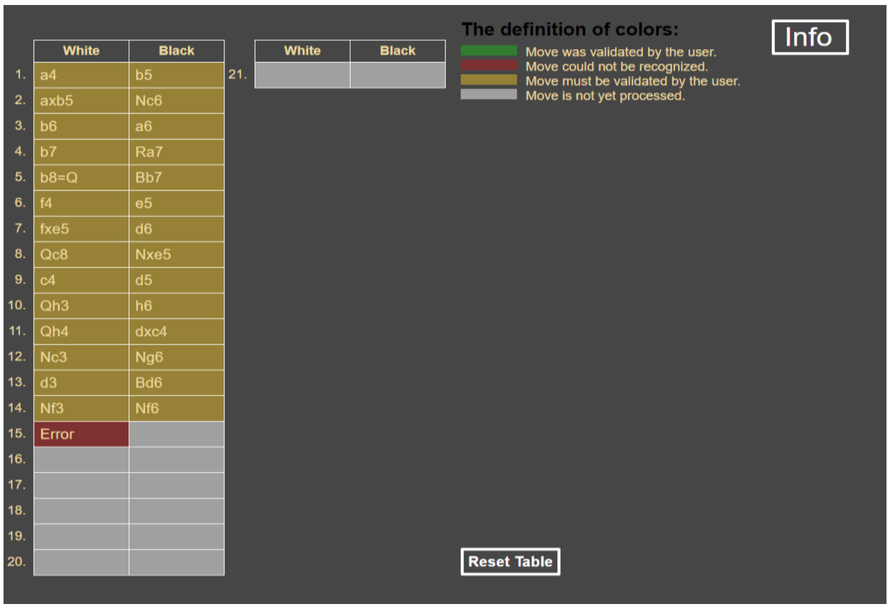
Extracting your moves...

© The Regency Chess Company 2012 e-mail info@regencychess.co.uk http://www.regencychess.co.uk

Figure 12 The loading screen that is displayed in every asynchronous step.

## 4.3 Output/replay

Once the moves are processed, the prediction is presented to the user in a table with the same layout as given on the scorecard.



| | White | Black |
|---|---|---|
| 1. | a4 | b5 |
| 2. | axb5 | Nc6 |
| 3. | b6 | a6 |
| 4. | b7 | Ra7 |
| 5. | b8=Q | Bb7 |
| 6. | f4 | e5 |
| 7. | fxe5 | d6 |
| 8. | Qc8 | Nxe5 |
| 9. | c4 | d5 |
| 10. | Qh3 | h6 |
| 11. | Qh4 | dxc4 |
| 12. | Nc3 | Ng6 |
| 13. | d3 | Bd6 |
| 14. | Nf3 | Nf6 |
| 15. | Error | |
| 16. | | |
| 17. | | |
| 18. | | |
| 19. | | |
| 20. | | |

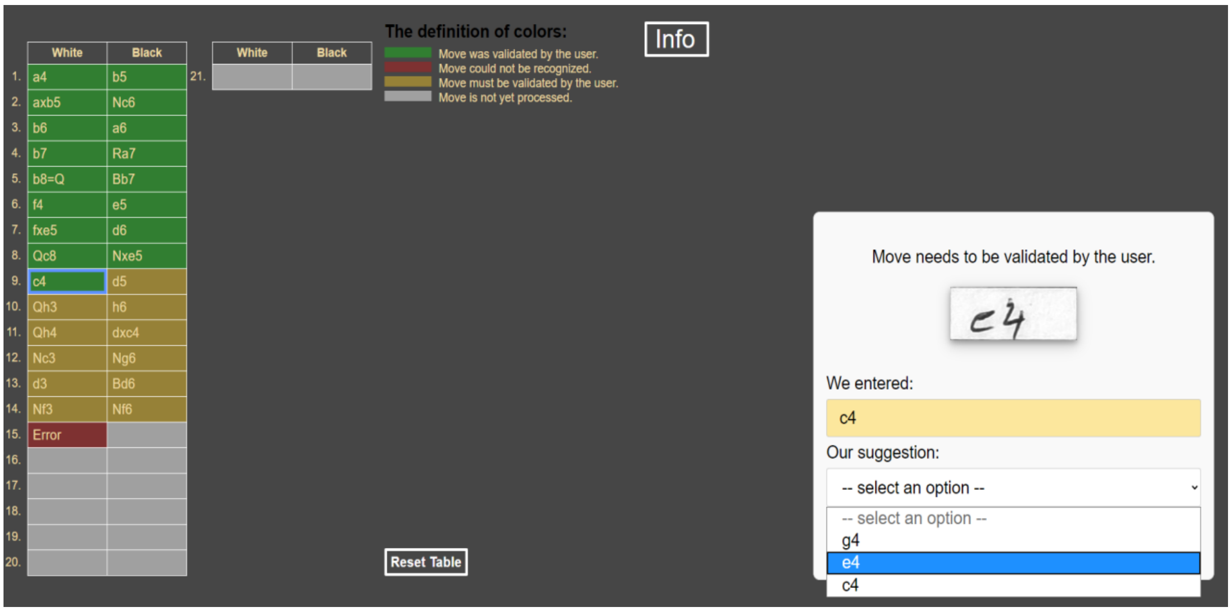
| | White | Black |
|---|---|---|
| 21. | | |

The definition of colors:

| | |
|---|---|
| 🟩 | Move was validated by the user. |
| 🟥 | Move could not be recognized. |
| 🟨 | Move must be validated by the user. |
| ⬜ | Move is not yet processed. |

Figure 13 The predicted game that is presented to the user based on the recognition.

The user has the possibility to control a prediction by clicking on one. A popup window displays an image of this box and the predicted move as shown in Figure 14. A every entry in the table has a colour and every colour represents a different state (Figure 15). For example, if the move is highlighted in green it corresponds to "Move was validated by the user" and it can be regarded as the true and thus correct move for this specific recognition (for this specific move). This state is reached by either changing the value or selecting one of the suggested candidates from the drop-down list. The correction can be applied with the "Apply" button.



|  | White | Black |
|---|---|---|
| 1. | a4 | b5 |
| 2. | axb5 | Nc6 |
| 3. | b6 | a6 |
| 4. | b7 | Ra7 |
| 5. | b8=Q | Bb7 |
| 6. | f4 | e5 |
| 7. | fxe5 | d6 |
| 8. | Qc8 | Nxe5 |
| 9. | c4 | d5 |
| 10. | Qh3 | h6 |
| 11. | Qh4 | dxc4 |
| 12. | Nc3 | Ng6 |
| 13. | d3 | Bd6 |
| 14. | Nf3 | Nf6 |
| 15. | Error |  |
| 16. |  |  |
| 17. |  |  |
| 18. |  |  |
| 19. |  |  |
| 20. |  |  |

|  | White | Black |
|---|---|---|
| 21. |  |  |

The definition of colors:

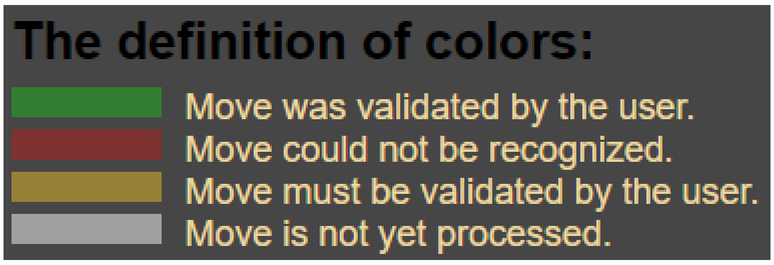
🟩 Move was validated by the user.
🟥 Move could not be recognized.
🟨 Move must be validated by the user.
⬜ Move is not yet processed.

Move needs to be validated by the user.

We entered: c4

Our suggestion: -- select an option --

Figure 14 A popup window for the blue framed move "c4" in the table. The window shows the original image, what was predicted by the algorithm and the top suggestions. This move must be corrected to "e4".

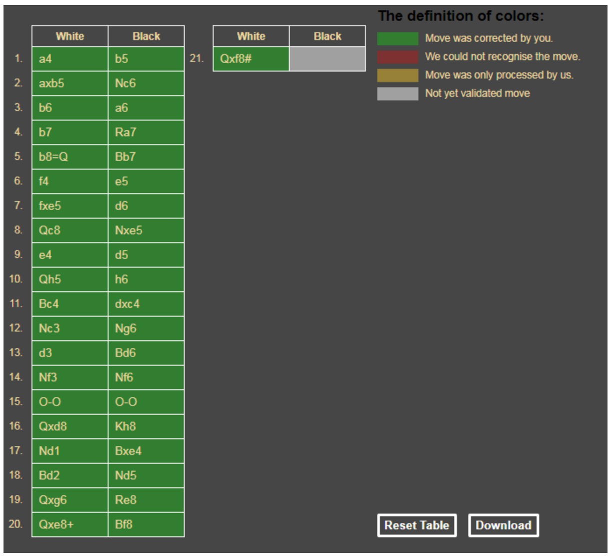


The definition of colors:

🟩 Move was validated by the user.
🟥 Move could not be recognized.
🟨 Move must be validated by the user.
⬜ Move is not yet processed.

Figure 15 The possible colours and their meaning in the frontend PGN table.

If a correction is applied, the table data is sent to the server and processed with the corrections. Meanwhile the loading screen is shown. This process is repeated until all predictions are correct. If the user wants to reset everything to the original prediction, he can do so by pressing the "Reset Table" button. If everything is correct, the "Download" button appears as show in Figure 16.



| | White | Black |
|---|---|---|
| 1. | a4 | b5 |
| 2. | axb5 | Nc6 |
| 3. | b6 | a6 |
| 4. | b7 | Ra7 |
| 5. | b8=Q | Bb7 |
| 6. | f4 | e5 |
| 7. | fxe5 | d6 |
| 8. | Qc8 | Nxe5 |
| 9. | e4 | d5 |
| 10. | Qh5 | h6 |
| 11. | Bc4 | dxc4 |
| 12. | Nc3 | Ng6 |
| 13. | d3 | Bd6 |
| 14. | Nf3 | Nf6 |
| 15. | O-O | O-O |
| 16. | Qxd8 | Kh8 |
| 17. | Nd1 | Bxe4 |
| 18. | Bd2 | Nd5 |
| 19. | Qxg6 | Re8 |
| 20. | Qxe8+ | Bf8 |

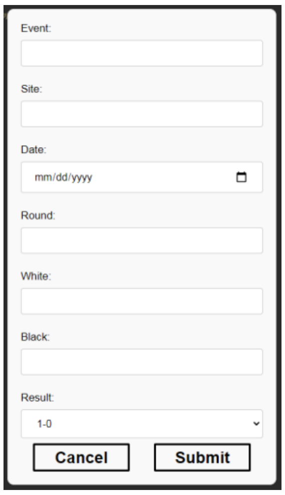
| | White | Black |
|---|---|---|
| 21. | Qxf8# | |

The definition of colors:

| | |
|---|---|
| 🟩 | Move was corrected by you. |
| 🟥 | We could not recognise the move. |
| 🟨 | Move was only processed by us. |
| ⬜ | Not yet validated move |

Figure 16 A correct game which is successfully predicted and validated by the user.

By pressing the "Download" button, the user is presented with a form which he can optionally fill in (Figure 17). The form can be filled with metadata about the players, the tournament, and the winner to complete the PGN file. The user can confirm this form with the "Download" button to download a PGN file with the entered information and the processed game.



Event:

Site:

Date: mm/dd/yyyy

Round:

White:

Black:

Result: 1-0

Figure 17 The final form to fill in the meta data for the PGN header.

# 5 Implementation

In the following chapter, the implementations of the image processing and text recognition algorithms are described as it was introduced in the Methods and procedures. It is split into six subchapters: Image alignment, Table extraction, Chess move localisation, CNN approach, ICR engine implementation and Improvement of recognition. All of those contain individual parts of the algorithm. In each chapter the results are shown, and further optimizations are discussed. The implementation was either tested on the layout data set, containing six different scorecard layouts or on the recognition data set containing 21 written scorecards.

## 5.1 Image alignment

To identify the tables in subsequent steps of the algorithm, the images that are uploaded must always be aligned straight. Since this cannot be presupposed, it is solved automatically. This ensures a constant starting position for the subsequent steps.

An already existing solution, the mobile document scanner [5], is used to align images. The script requires as a source an image of a sheet of paper on which all four corners are visible and outputs an aligned grayscale image with the background excluded. This procedure is shown in the image below. The source code is integrated into the existing pipeline with the adaptation that the source image is already in grayscale and not full colour.

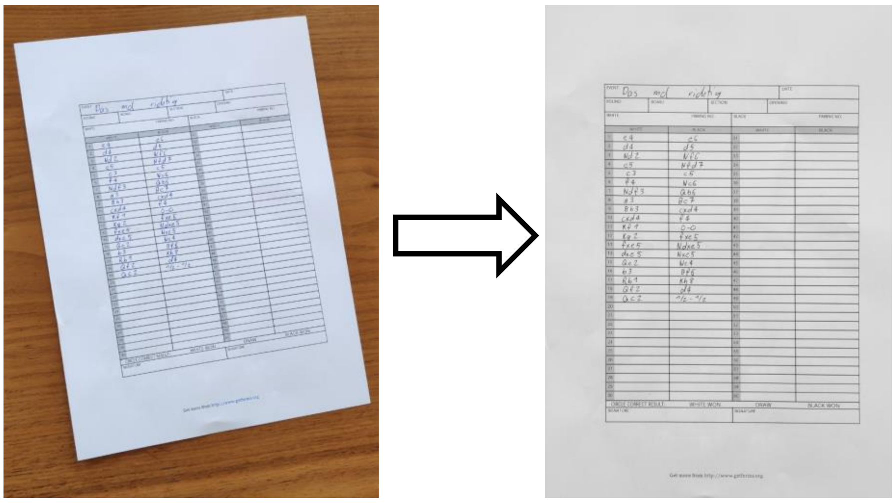


Figure 18 Shows the original image before the alignment on the left and after the aligned image on the right.

### 5.1.1 Results & Discussion

The scanner solves the initial problem, that the user's images are not perfectly aligned. It also recognizes the perspective of an image and corrects it to a top-down view. This is exclusively possible if the scorecard can be distinguished from the background and all four vertices of the scorecard are clearly visible. In this case, the image can be transformed.

The image alignment could be further improved by optimizing the script and filters. Image distortions exist which the scanner cannot correct, for example waves in the paper. Furthermore, there are image quality requirements that must be met. If the quality of the image is too low, the alignment will fail, and the image must be retaken with better quality. Moreover, the vertices are not always found due to the background being reflective or too bright.

Another possible approach would be to use feature-based image alignment[12]. To do so, an aligned source image is compared to the input image with the same table format and based on its features the input image is mapped to the source image. The problem of such an approach is that a standardized scorecard would have to be published or users would have to upload aligned source images into a database. Those uploaded scorecards could then be used as reference for comparison. Also, the same problem arises, since aligned source images are a must and they can have errors in their alignment.

[12] Feature based image alignment.
Available : https://sites.google.com/site/imagealignment/tutorials/feature-based-vs-direct-image-alignment [05.06.2020]

## 5.2 Table extraction

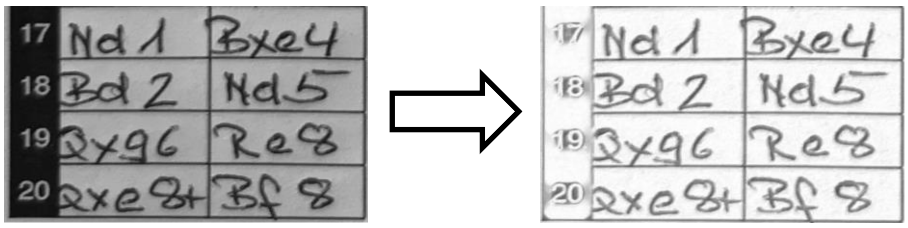
From the aligned scorecard the tables are extracted. Today's state of the art for written data extraction in a document considers two approaches. Either the algorithm searches and orients itself by the table lines or ignores the table lines and searches for text regions which are arranged in a table structure [6]. In this thesis the search for table lines seemed more promising and thus this approach was chosen.

Since the image was aligned in the previous step, it can be assumed that the table lines are horizontal and vertical. The pre-processing was inspired and built upon a blog post [7] presenting a pre-processing solution applied to a similar starting scenario. Only Python, OpenCV and NumPy functions were used for the procedures below. To spot the details in the images, all following figures are zoomed into the bottom left area of an example scorecard which fits the requirements of the application.

### 5.2.1 Pre-processing

A series of consecutive steps must be followed to correctly pre-process the image and enables a reliable table extraction. Following, the steps are described in ascending form.

**1. Shadow removal:** Through online research, a way to remove shadows in grayscale images was searched and implemented. This step helps to improve the later removal of noise and to clearly separate the writing from the background. In Figure 19, the shadow removal is applied. As can be seen, small gaps are created between the "Q" and the black numbering area.



| | | |
|---|---|---|
| 17 | Nd 1 | Bxe4 |
| 18 | Bd 2 | Nd5 |
| 19 | Qxg6 | Re8 |
| 20 | Qxe8+ | Bf 8 |

Figure 19 The left image shows the source image of a table part. Shadow removal is applied to the right image.

**2. Adaptive threshold:** With the aim of obtaining a binary image that clearly separates the table from the background, a binary threshold is used. This threshold can be applied either globally or adaptively. Global thresholding uses a single value. The pixel values greater than the defined value become 1 and the others become 0. In adaptive thresholding, the threshold value changes depending on the area of the image. This is beneficial if an image has different lighting conditions or shadows. The difference can be clearly seen in Figure 20.

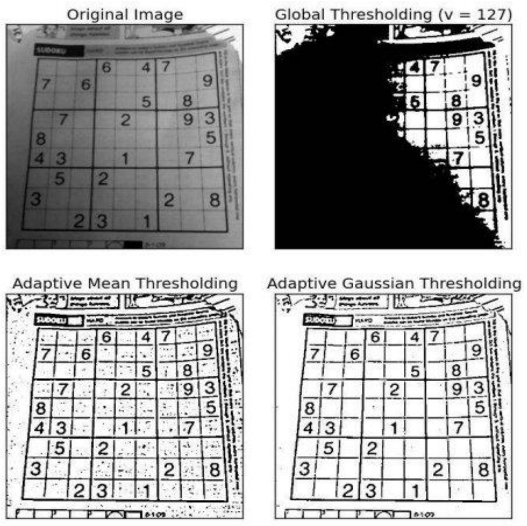

Original Image
Global Thresholding (v = 127)
Adaptive Mean Thresholding
Adaptive Gaussian Thresholding

Figure 20 Different thresholding methods applied to a sudoku sheet.

Despite the previously applied shadow remover, images might still have different lighting conditions. For this reason, a binary adaptive mean threshold is used. The small gaps created by the shadow removal lead to bigger gaps in the table lines, displayed in Figure 21.

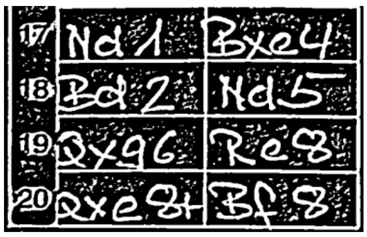


Figure 21 Binarized part of the table with gaps in the left table line.

**3. Morphological operations:** To fill the gaps in the table lines, additional steps must be taken. Morphological operations are used to extract long horizontal and vertical lines from the image. These operations use kernels as an operating tool. A kernel is a matrix containing binary values. The morphological kernels used are a "1 x n" kernel for the vertical lines and "n x 1" kernel for the horizontal lines with n pixels (Figure 22).

```
[[1],
 [1],
 [1],                    [[1],[1],[1],[1],[1]]
 [1],
 [1]]
```

Figure 22 Morphological kernel for vertical and horizontal lines with a length of 5 pixels.

During erosion, if any pixel underneath the form is not 1, all these pixels are set to 0. During dilation, all pixels underneath the form are set to 1 if at least one of them is 1. Combinations of these operations are very useful to fill gaps in lines, reduce noise or to filter specific shapes in an image [8]. Therefore, erosion removes all pixels that do not form a line as long as the kernel form and dilation expands all pixels by a length equal to the kernel form.

To extract the table, the image is once morphologically reduced into the horizontal and once into the vertical lines. The following examples just show the vertical application of such morphological operations. The same procedure is applied to get the horizontal lines.

First, a small kernel is defined which removes all minor noise with one erode and fills all gaps in the table lines with two dilates. For this, the kernel must be larger than the noise contained in the image after the adaptive threshold is applied. Otherwise, a single erosion would not remove any noise and thus results in the upcoming dilation extending the noise into one connected line. In any case, the kernel is chosen so small that no separated table parts are lost by the single erode. These small table parts containing gaps can be seen in the left image in Figure 23. Due to the small kernel, not all the noise is extended into a big line and only the real table lines are extended. Moreover, the kernel must be chosen small enough that no text fragments merge into a line due to the doubled dilation. This problem is shown in Figure 23 on the right side. All the noise in the left image is extended, as well as the table lines. However, the noise can still be separated from the table lines due to its length in the image. With a bigger kernel, the noise would form a long straight line and the table recognition would fail. The kernel is adapted with trial and error until all cleanly photographed images work. In the end, the horizontal kernel is set to a relative size of 1/200 of the image width and the vertical kernel to 1/100 of the image height.

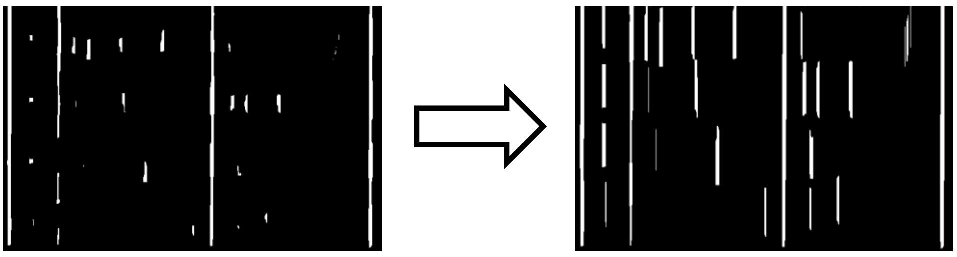


Figure 23 The left image is the table part after one erosion and the right image after two dilations with a small kernel.

The next step is to define a large kernel, which removes all lines that are too small to represent a table guideline with a single erode and dilate. The longer this kernel is chosen, the less tolerance the table lines have in their slanted position. Otherwise even slightly slanted lines are removed since they do not match the kernel in its full height or width. Therefore, the kernel should be chosen as small as possible. Thus, the horizontal kernel is set to 1/20 of the image width and the vertical kernel to 1/10 of the image height.

In Figure 24, the output after a single erode and dilate can be seen. It shows that only long lines are left, and all the noise is removed.

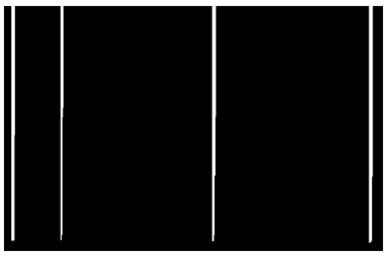


Figure 24 The table lines left after a single erosion and dilation with a large kernel.

Slightly slanted lines may lead to gaps within vertical lines (Figure 25). To fill these a square kernel with a single dilation and erosion is applied. This kernel is set to the size 5x5, because with this initial size the problem is solved.

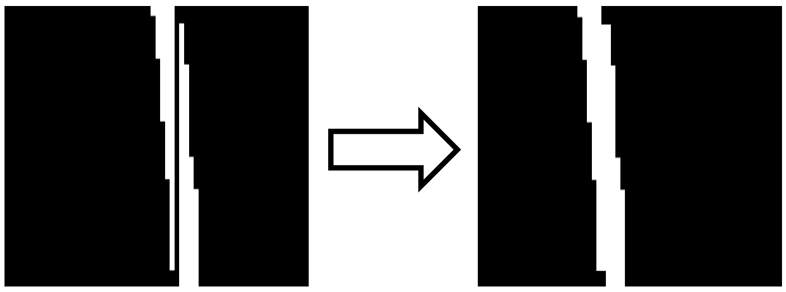


Figure 25 The left image shows a slanted table line which is split. The right image shows the same line after a single erosion and dilation with a square kernel.

To combine the vertical and horizontal lines, both images are merged. In some cases, the lines do not fully connect, thus a square kernel is used to dilate the image twice. This gives thicker lines and corrects slight errors at the intersection (Figure 26).

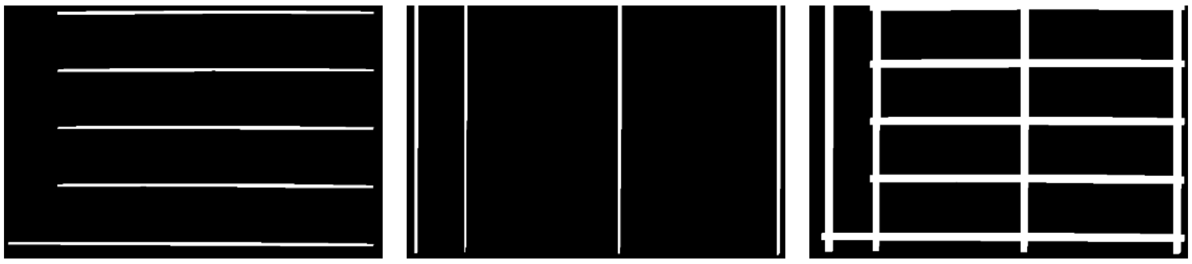


Figure 26 The right table image is the combination of the other two images which contain all horizontal and all vertical table lines. Also, dilation with a 5x5 kernel is applied.

### 5.2.2 Results & Discussion

Once the whole procedure is applied, the image is inverted which leaves only the extracted table from the scorecard (Figure 27). This procedure for table extraction generally works reliably for correctly aligned and focused images in high resolution. Thanks to the use of dynamic kernel sizes, different image resolutions are supported.

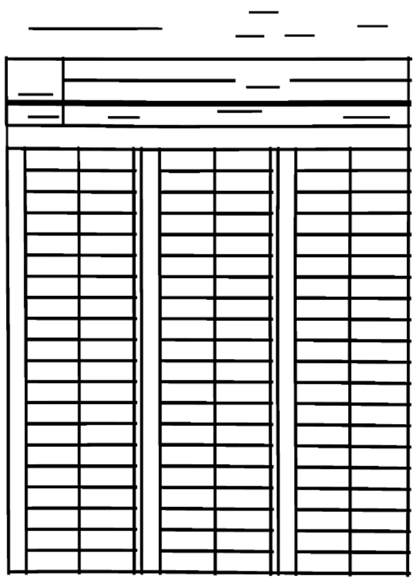


Figure 27 Binary image of the scorecards table after the table extraction.

Layouts with dotted lines or dashed lines are also examined. However, no reliable line detection is achieved. Dotted lines behave very similar like noise and get filtered out with the noise, whereas dashed lines are partially detected and filled but have too many gaps to yield a fully connected table.

As seen in Figure 26 it could be approximated which lines must be of equal length. Such correlations could help to improve detection for shortened or interrupted lines and could be corrected by the algorithm.

The implemented kernels have a size relative to the input image. It is possible that kernels adapting linearly with image resolution may yield improved reliability and should be investigated in future research.

## 5.3 Chess move localisation

In the previous step a binary table was created. From this, the zones in which the player writes his chess moves from the original picture must be detected. In the following, the rectangles found in the binary image which represent the zones that contain writing, will be referred to as “boxes” and the “relevant boxes” represent the “true positives” that contain the actual chess moves.

### 5.3.1 Box detection

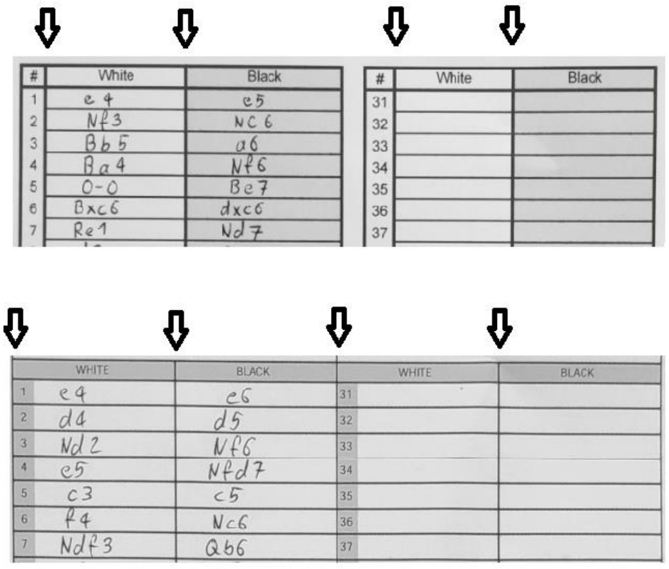
In this step contours and minimum bounding rectangles are used to locate the chess moves. A contour may be best described as a curve that connects all continuous pixels with the same intensity or colour. This will surround every shape in an image and can be further used for object recognition or position detection of said object. The OpenCV function "cv.findContours()" is used to identify all contours in the image shown in Figure 27 with the parameter “chain_approx_simple” an array of contour vertices is returned. The array can be passed to the function "cv.boundingRect()" which encloses a given contour with the smallest possible rectangle, a bounding rectangle. Subsequently this detection algorithm is explained in more detail.

**1. Anchor line detection:** In a table, the boxes are ordered into columns. To the left and right of such a column are long vertical lines. These vertical lines are defined as anchor lines and used to locate the adjacent boxes.

Since the previous step separated vertical and horizontal lines from the image, the vertical lines can be reused. From these, only the lines which are at least as high as one third of the image are considered. Out of all these, exclusively the ones on the left of the move boxes are selected as anchor lines. As shown the upper image of Figure 28, there are eight possible anchor lines but only four are selected. In the lower image, no spacing is present and thus the lines are selected normally.

Additionally, to filter the row number lines, any lines that are too close together are removed. For this purpose, the average distance between all vertical table lines is calculated. The program runs from left to right and discards any line that is closer to its right neighbour than the average distance plus or minus a threshold of 1/60 of the picture width. The threshold has been chosen through with trial and error until it worked successfully for all available layouts in the layout data set.

The x-axis values of the selected anchor lines are stored to later sort the boxes by their x-coordinates. Additionally, vertical lines that are too close to the left or right edge of the image are considered incorrect, as they are created by the image alignment if the scorecard does not lie flat on the surface.



| # | White | Black |
|---|---|---|
| 1 | e4 | e5 |
| 2 | Nf3 | NC6 |
| 3 | Bb5 | a6 |
| 4 | Ba4 | Nf6 |
| 5 | O-O | Be7 |
| 6 | Bxc6 | dxc6 |
| 7 | Re1 | Nd7 |

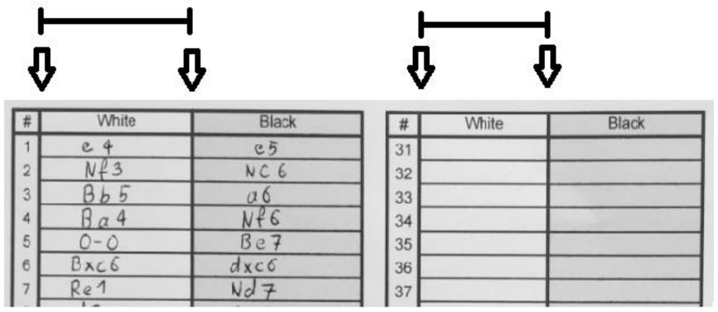
| # | White | Black |
|---|---|---|
| 31 | | |
| 32 | | |
| 33 | | |
| 34 | | |
| 35 | | |
| 36 | | |
| 37 | | |

| | WHITE | BLACK | | WHITE | BLACK |
|---|---|---|---|---|---|
| 1 | e4 | e6 | 31 | | |
| 2 | d4 | d5 | 32 | | |
| 3 | Nd2 | Nf6 | 33 | | |
| 4 | e5 | Nfd7 | 34 | | |
| 5 | c3 | c5 | 35 | | |
| 6 | f4 | Nc6 | 36 | | |
| 7 | Ndf3 | Qb6 | 37 | | |

Figure 28 Relevant anchor lines on two different table layouts.

**2. Assigning boxes to anchor lines:** Based on the found anchor lines, all the bounding rectangles are now compared against the final set of anchor lines by their x-coordinates. A bounding rectangle is regarded as valid if its deviation from the anchor lines x-coordinate is smaller than the threshold of 1/80 of the image width. This threshold was defined after validating its function on the whole layout data set. After all boxes are assigned, they are sorted in order of their y-coordinate to regain the structure of the table.

**3. Filtering boxes:** Since all remaining boxes are only filtered by their x-coordinate, there might still be faulty boxes within the valid coordinates. It is now a matter of locating the boxes containing the chess moves. For this step it is assumed that all boxes containing the chess moves have a uniform size per column. It must be evaluated per column, since it is not always the case that the boxes have the same width over the whole scorecard (Figure 29).



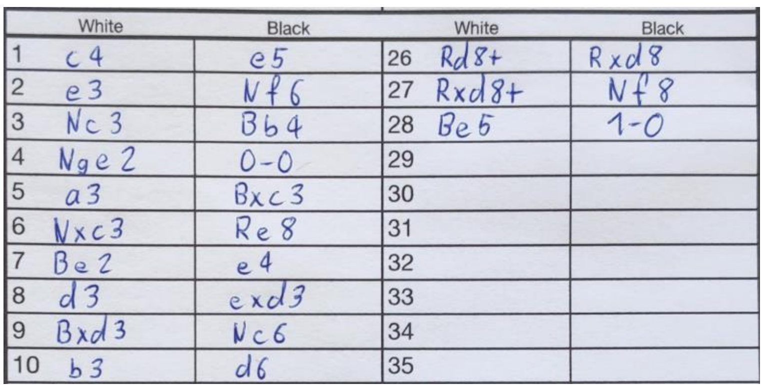
| # | White | Black |
|---|---|---|
| 1 | e4 | e5 |
| 2 | Nf3 | Nc6 |
| 3 | Bb5 | a6 |
| 4 | Ba4 | Nf6 |
| 5 | 0-0 | Be7 |
| 6 | Bxc6 | dxc6 |
| 7 | Re1 | Nd7 |

| # | White | Black |
|---|---|---|
| 31 | | |
| 32 | | |
| 33 | | |
| 34 | | |
| 35 | | |
| 36 | | |
| 37 | | |

Figure 29 The widths of columns are not equal. The two marked widths show clear differences.

To locate the most prominent box (bounding rectangle) property, the median can be used regarding the width and width-height ratio. The width is used since it is always larger than the height and consequently inaccurate box contours have smaller error percentages when compared using the height. To validate the selected boxes, the width-height ratio is calculated to only select boxes that fit the ratio and not only the width. Because of the slight deviations caused by image distortion or image alignment, the width deviation threshold is set to 5% and the width-height deviation threshold to 15%.

**4. Removal of row enumerations:** If the table contains an enumeration, these must be removed. This is necessary since the recognition of ABBYY results in a worse output if the numbers are left inside the boxes. It is assumed that the number enumeration is located to the left of the moves and that the enumeration has a uniform size and positioning throughout the table (Figure 30). If no enumeration exists, this part of the algorithm will not alter anything.



| White | Black | White | Black |
|---|---|---|---|
| 1 c4 | e5 | 26 Rd8+ | Rxd8 |
| 2 e3 | Nf6 | 27 Rxd8+ | Nf8 |
| 3 Nc3 | Bb4 | 28 Be5 | 1-0 |
| 4 Nge2 | 0-0 | 29 | |
| 5 a3 | Bxc3 | 30 | |
| 6 Nxc3 | Re8 | 31 | |
| 7 Be2 | e4 | 32 | |
| 8 d3 | exd3 | 33 | |
| 9 Bxd3 | Nc6 | 34 | |
| 10 b3 | d6 | 35 | |

Figure 30 An example of a table layout containing enumeration in the move boxes.

To locate the enumeration, the sector where the enumeration is suspected must be recognized with an OCR engine. Python-tesseract[13] is used to achieve this which is a wrapper for Google's Tesseract-OCR engine. It recognizes printed writing, offers different settings and supports all popular image file standards. Since it can be installed and run locally, the implementation is faster than ABBYY. Furthermore, the OCR is applied iteratively which makes API calls even slower based on the connection establishment and queue of ABBYYs process.

To determine if an enumeration exists, the last ten move boxes are selected. They are usually empty and contain the highest numbers which are assumed to be the widest. Only the left third of each box is inspected since the enumeration is expected to be on the left side. Since not the whole box is recognized, the computational time needed for the iterative application is reduced. The output of the recognition is searched for an enumeration pattern which is further explained in the next step. The inspected image parts are halved if an enumeration is found and this process is repeated until no more pattern is found. Eventually the enumeration pattern is broken. Once the pattern breaks, the width of the last working iteration (n-1) is used to remove the numbers in all boxes.

### Searching of enumeration in the recognition

For each box, the recognition string of "python-tesseract" is converted into positive numbers and put into an array. Then every number is compared to each other number in the array. If the index distance between any two numbers are equal to the difference of the numbers, they are in

[13] Pytesseract project description. Available: https://pypi.org/project/pytesseract/ [15.04.2020]

enumerating order and the matching counter is increased. The image below shows an example of an array with five numbers where four are in order.

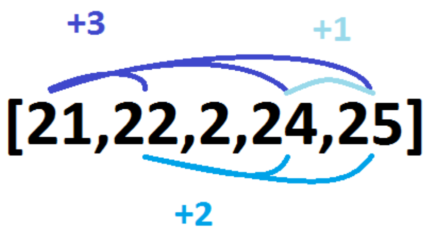
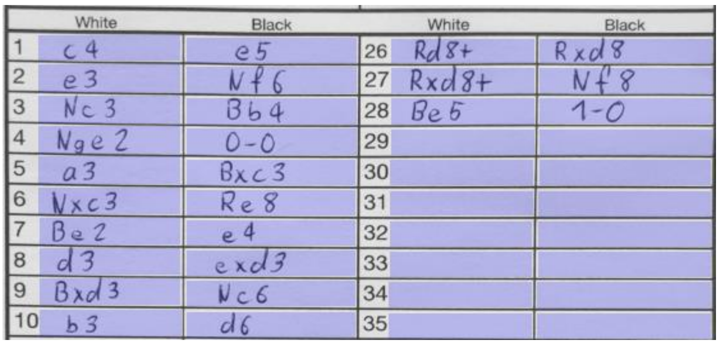


Figure 31 Example of the counting of numbers that are in order.

This approach of counting for an enumeration behaves like a Gaussian sum. This counting method has the advantage that multiple enumerations can be distinguished from each other. For example, if ten out of ten moves are in order [0,1,2,3,4,5,6,7,8,9], the matching count is 55. If eight out of ten moves are in one order and the other two moves are in another order [0,1,2,3,4,5,6,7,0,1], the matching count is then 37, which concludes that not all moves are in the same order.

Since the OCR has recognition flaws, a number pattern is found if 60% of the numbers are in order, which corresponds to a matching count of 21. This percentage was chosen through trial and error and with the intention to support as many recognition flaws as possible. The blue boxes in Figure 32 show the final area selection per box.



| | White | Black | | White | Black |
|---|---|---|---|---|---|
| 1 | c4 | e5 | 26 | Rd8+ | Rxd8 |
| 2 | e3 | Nf6 | 27 | Rxd8+ | Nf8 |
| 3 | Nc3 | Bb4 | 28 | Be5 | 1-0 |
| 4 | Nge2 | 0-0 | 29 | | |
| 5 | a3 | Bxc3 | 30 | | |
| 6 | Nxc3 | Re8 | 31 | | |
| 7 | Be2 | e4 | 32 | | |
| 8 | d3 | exd3 | 33 | | |
| 9 | Bxd3 | Nc6 | 34 | | |
| 10 | b3 | d6 | 35 | | |

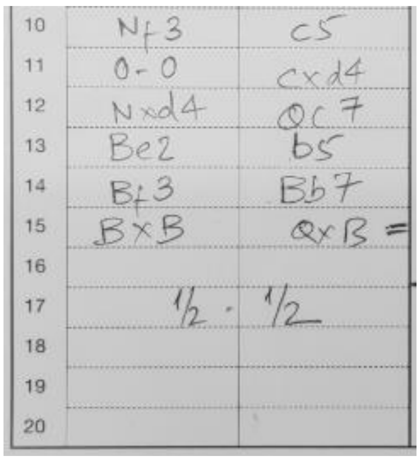
Figure 32 Marked in blue are the final boxes. The leading numbers are cut off.

**5. Sorting of boxes:** In the last step of the chess move detection the relevant boxes must be re-sorted before passing the data to the next steps. They are sorted in the same order as they were played in a chess game. Using the x- and y-coordinates of the relevant boxes. It is also recorded how many moves were in each column so that the same layout can be displayed at a later stage.

### 5.3.2 User validation of relevant boxes

Since solely the boxes containing the written moves are relevant, all other boxes must be removed. In practice results or signatures are written over the writing region as can be seen in Figure 33. Thus, all boxes after the lastly written chess move are irrelevant. To locate this box a user input is used.

If any of the remaining boxes does not contain a written chess move, the user is able to deselect them in a follow up step. This selection process was previously shown in chapter 4.2.



| | | |
|---|---|---|
| 10 | Nf3 | c5 |
| 11 | 0-0 | cxd4 |
| 12 | Nxd4 | Qc7 |
| 13 | Be2 | b5 |
| 14 | Bf3 | Bb7 |
| 15 | BxB | QxB = |
| 16 | | |
| 17 | ½ · | ½ |
| 18 | | |
| 19 | | |
| 20 | | |

Figure 33 A scorecard where the result is written into the move boxes.

### 5.3.3 Results & Discussion

Using the algorithm described above, all relevant boxes are detected. The algorithms are chosen in such a way that they work in all table layouts if the table has solid lines. To localize the chess moves, the orientation by the table lines is implemented.

With this said, problems may still arise if the relevant boxes contain other boxes than just the ones containing the chess moves. As boxes are only filtered regarding their median properties, errors are introduced when other boxes have the same properties and the same x-coordinate. This problem is solved by extending the web interface to take user input to validate the found boxes. This procedure could be partially automated since the move boxes are usually in a distinct pattern and not spaced apart or scattered. However, if the faulty boxes fit into the pattern of the move boxes (for example "Black" and "White" at the top of the column), one must rely on the ICR engine display the true content of those. With the current approach of fuzzy string matching, strings that are nowhere near SAN syntax receive a heuristic score close to zero. This indicates that the found box might not belong to the chess moves and could be removed when further improving the algorithm and usability.

Another challenge arises from an unsuccessful table extraction which leads to unconnected table lines and consequently to wrong or missing boxes. This problem is illustrated in Figure 34 and could be fixed with adding the missing boxes based on the surrounding pattern.

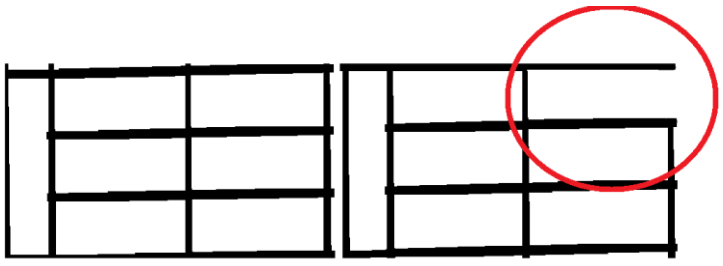
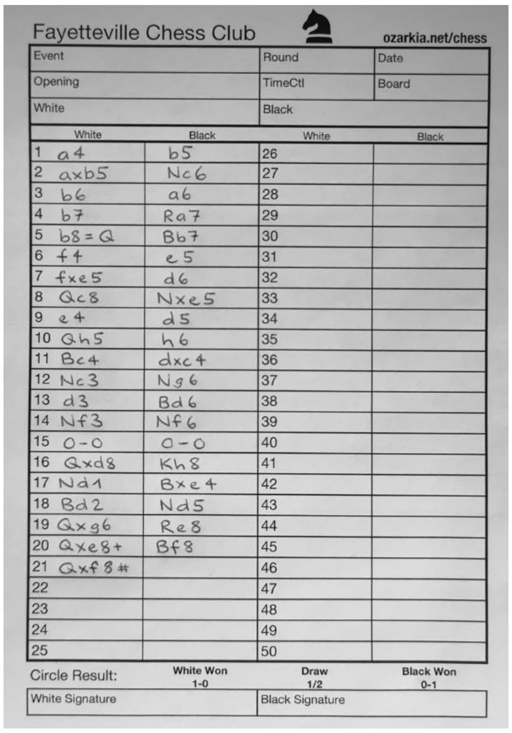


Figure 34 Scorecard with a gap in the top right corner, which is produced by the pre-processing.

To validate that the removal of leading numbers is working correctly and that the removal improves the recognition a test is conducted with a scorecard that contains row numbering inside the move boxes. This test image is a best-case scenario. It is not blurry, aligned properly and the writing is completely inside the boxes and does not reach over the borders (Figure 35).



Fayetteville Chess Club ozarkia.net/chess

| | | |
|---|---|---|
| Event | Round | Date |
| Opening | TimeCtl | Board |
| White | Black | |

| White | Black | White | Black |
|---|---|---|---|
| 1 a4 | b5 | 26 | |
| 2 axb5 | Nc6 | 27 | |
| 3 b6 | a6 | 28 | |
| 4 b7 | Ra7 | 29 | |
| 5 b8=Q | Bb7 | 30 | |
| 6 f4 | e5 | 31 | |
| 7 fxe5 | d6 | 32 | |
| 8 Qc8 | Nxe5 | 33 | |
| 9 e4 | d5 | 34 | |
| 10 Qh5 | h6 | 35 | |
| 11 Bc4 | dxc4 | 36 | |
| 12 Nc3 | Ng6 | 37 | |
| 13 d3 | Bd6 | 38 | |
| 14 Nf3 | Nf6 | 39 | |
| 15 0-0 | 0-0 | 40 | |
| 16 Qxd8 | Kh8 | 41 | |
| 17 Nd1 | Bxe4 | 42 | |
| 18 Bd2 | Nd5 | 43 | |
| 19 Qxg6 | Re8 | 44 | |
| 20 Qxe8+ | Bf8 | 45 | |
| 21 Qxf8# | | 46 | |
| 22 | | 47 | |
| 23 | | 48 | |
| 24 | | 49 | |
| 25 | | 50 | |

Circle Result: White Won 1-0 Draw 1/2 Black Won 0-1

| | |
|---|---|
| White Signature | Black Signature |

Figure 35 Evaluation scorecard with enumeration in the left move boxes.

The test is performed using two different regular expressions and lettersets for ABBYY. The full test description can be found in the appendix in chapter 9.5.1.

Problems are encountered if the scorecard contains enumerations inside the boxes. The recognition results in errors, no matter if the regex and the letterset are adjusted or not. Numbers like 2, 6, and 9 among others are not reliably detected and it is not possible to define a general solution that works for all cases. Furthermore, some numbers are interpreted as completely different character combinations such as “11” as “K8” which does not aid the interpretation of the recognized move.

## 5.4 CNN approach

Once all relevant boxes have been found, they must be processed by an ICR engine. Since custom CNNs are very efficient and specialized, they are most promising. The big advantage compared to the alternatives is that they can be trained to the problems needs. This training allows for a specialized alphabet and optimization. It is evident that this produces good results, which “Reine - Chess" has proven in their application. The following steps show how the individual characters are extracted from a move box and how to prepare them for a CNN.

### 5.4.1 Extracting single characters

MSER is used for blob detection in images. MSER stands for “maximally stable extremal regions” and the goal of blob detections is to detect regions in images that differ in brightness or colour compared to their surrounding region. With MSER this is achieved by calculating binary regions with different thresholds and keeping the regions which are persistent over several threshold values [9]. This method is widely used for automatic text recognition in natural images [10]. Since this approach has produced reasonable results from the beginning, no other approaches are considered.

**Pre-processing**

MSER is very sensitive to noise. To reduce noise and fill up pixel errors, the image is first binarized with an OTSU threshold and then blurred with a Gaussian kernel to smooth the edges of the characters (Figure 36). Since the image background is mostly in the same grayscale, OTSU threshold dynamically searches the best value to binarize each image.

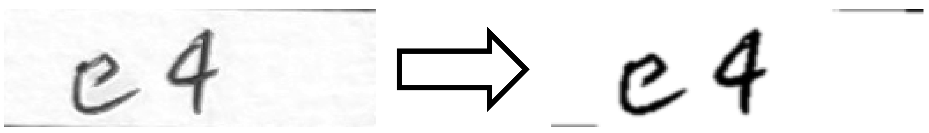


Figure 36 On the left side is the original image of a handwritten move in a box. The right image is binarized and blurred.

MSER is then used to detect pixel regions that are persistent over several threshold values. This creates boxes which are not only around the characters, but every pattern that differs from the background. As can be seen in Figure 37, boxes are created outside the characters at the upper and lower image boundaries by the remains of the table, as well as a large box which surrounds the entire image. Boxes are also created within the letters as can be seen inside the character four in the image below. The thick lines surrounding the characters are several boxes created by MSER.

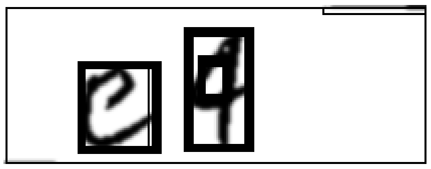


Figure 37 All created boxes by MSER drawn over the pre-processed image.

To find the correct box for each character out of all possible MSER boxes, three filter methods are applied in the following order:

**1. Area boundary:** Since the characters always occupy a certain area in the image, boxes that are too large or too small can be excluded with a minimum and maximum area threshold. To determine these threshold values, two scorecard layouts are considered, see Figure 38. In the left image, writers tend to use more of the available space than in the right image.

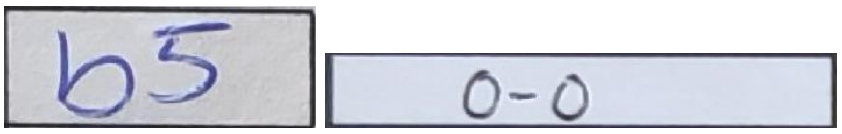


Figure 38 Two different move boxes on which the writers use different amounts of the available space.

Based on the different layouts and the application to various scorecards, the maximum area of the box to be cut is set to 1/3 and the minimum area to 1/500 of the total area. With the minimum area threshold all boxes smaller than the smallest character, the hyphen, are removed. The MSER fields remaining after applying the minimum and maximum area thresholds are shown in Figure 39.

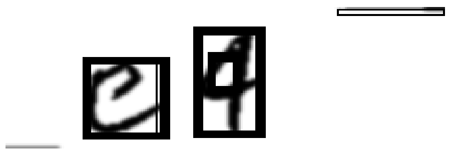


Figure 39 All remaining MSER boxes after applying the minimum and maximum area thresholds to the image.

**2. Border boundary:** As can be seen in the picture above, boxes can occur at the outer area of the picture. For that reason, two thresholds are defined, one for the horizontal and one for the vertical borders. After adapting them to various layouts and considering that the move boxes are always wider than high, the horizontal threshold is set to 5% of the box width and the vertical threshold to 20% of the box height. Thus, all boxes that are fully outside this area (red) are discarded, as shown in Figure 40.

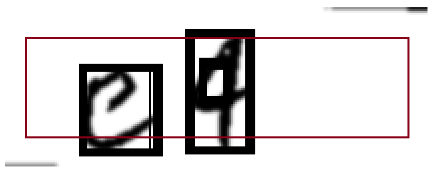


Figure 40 All remaining MSER boxes after applying the red marked area threshold to the image.

**3. Intersection boundary:** This leaves only the boxes detected by MSER that are located around and inside the real characters. The largest box is now selected and all boxes within the selected box are removed. For this purpose, each box is compared with each other and the percentage of overlapping area is compared. If the overlapping area is greater than 40% of the smaller box area, it is assumed that this box is part of the larger character and is discarded. This value is selected based on writing styles in which the writers write very close together and overlapping exists. The remaining boxes each surround one character as seen in Figure 41.

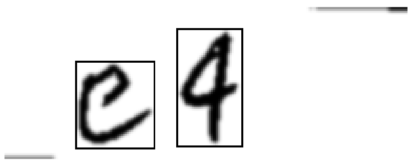


Figure 41 Situation after applying the intersection filter. Only the relevant boxes are left.

Finally, the characters are cut out based on the coordinates of the remaining MSER boxes from the original image seen in Figure 36.

### 5.4.2 EMNIST pre-processing

To enable later recognition with a CNN, the images must be pre-processed in the same way as the training data. The training data referenced is the EMNIST [4] data set. To pre-process the characters, the script from "Reine – Chess" GitHub repository[14] is used. EMNIST contains annotated 28x28 pixel images of digits and the alphabet in handwritten form. This is commonly used when classifying handwritten characters due to the size of annotated data. Figure 42 shows the pre-processed image after applying the algorithm.

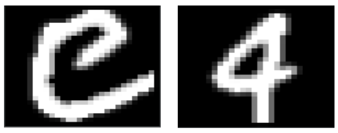


Figure 42 EMNIST pre-processing applied to the cut out characters.

[14] GitHub File of Reine – Chess. Available: https://github.com/Messier-16/Reine-Chess-Scoresheet-Scanner/blob/master/PreProcess.py [05.04.2020]

### 5.4.3 Results & Discussion

The region detection works reliably if the following conditions are met:

- The characters do not overlap any borders
- The characters do not overlap each other
- All parts of a character are connected to each other

Figure 43 shows that the region detection does not work if the above conditions are not satisfied.

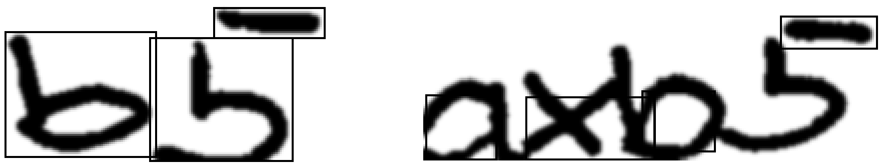


Figure 43 MSER region detection is applied to both images. The left image contains separated parts of a character and the right image contains characters that overlap with each other.

In conclusion, overlapping characters and disconnected character parts often occur in written scorecards. For this reason, the box detection must be improved or perhaps a different approach to MSER should be implemented. One example is edge enhanced MSER which is a combination of MSER and Canny edge detection [11].

The functions to cut out characters and pre-process them for a CNN are implemented. However, the approach was not pursued further for two reasons:

1. The handwriting is not spaced apart which makes it nearly impossible to extract every character by itself.
2. EMNIST does not contain special characters such as "+, =, #, - and /" which need to be gathered and annotated first.

The idea to annotate own characters was discarded since no alternative data set was found which contains the needed characters. Furthermore, the uncertainty if self-gathered and annotated data would work persisted. With the implemented steps a basis was created on which a self-trained CNN can be built and trained. However, the problems mentioned above should be solved first.

## 5.5 ICR engine implementation

Since the custom CNN approach was not continued an alternative was searched. This chapter first discusses the selection procedure of the final engine. Secondly, the implemented approaches are discussed, and the most difficult aspects are pointed out. Finally, the implementation of the final engine is evaluated to identify common recognition flaws and take advantage of them.

### 5.5.1 ICR engine selection

ICR engines can generally be divided into two categories. On one hand the recognition of text in a whole document and on the other hand the recognition of single text fields or characters. Furthermore, most services are API based and some are more expensive than others.

For our application, the ICR engine must support handwriting, also called handprinted writing. Only engines with this feature were considered in the selection process. Moreover, a custom alphabet and rules should be supported.

In the following sections multiple OCR engines and their features are listed and compared. For this selection, the most important factors are handwriting and zonal recognition support. The pricing was taken into consideration if the API's have similar specifications.

**ABBYY**

ABBYY is a web OCR service that provides full-page and zonal OCR through an API (Table 1) [12]. It supports the upload of images with specific region information. This helps to limit the recognition to only the specified fields and not the full document.

Table 1

Feature analysis of ABBYY.

| Feature | Description |
|---|---|
| Price | First 500 pages for free afterwards 30$ - 800$ per month.<br>100$ = 2'000 pages or 10'000 fields |
| Additional features | - Can limit the alphabet<br>- Supports regular expressions for chess logic<br>- Whole page input<br>- Writing styles (American, German, etc.) for different styles of writing i.e. 1 and 7. |

**LEADTOOLS ICR Module - OmniPage Engine**

LEADTOOLS ICR SDK is an ICR Module for .Net and C/C++ developers [13]. It cannot be used as a standalone application. It allows for zonal recognition and specification of all the parameters. It also combines pre-processing and OCR in one tool. However, it is not written in Python and offers no Python support.

Table 2

Feature analysis of LEADTOOLS ICR Module.

| Feature | Description |
|---|---|
| Price | Single license 3'000$. |
| Additional features | - Can limit the alphabet<br>- Huge documentation<br>- Zonal recognition support<br>- Automatic pre-processing |

**Google Cloud vision**

Google Cloud vision provides an API with a huge documentation and configurable requests [14]. However, the text recognition only supports full-page recognition which returns all recognized locations in the image with their content and their coordinates.

Table 3

Feature analysis of Google Cloud vision.

| Feature | Description |
|---|---|
| Price | First 1000 recognitions free, further recognitions are 1.50$ per 1000 units. |
| Additional features | - Can limit the alphabet<br>- Object/Facial recognition and other features supported that are not needed |

**Conclusion**

The examined services are API based and not free of charge. The API systems allow for handwriting support and can be integrated into the application. Google Cloud Vision was eliminated due to not supporting zonal recognition which means that the whole document must always be processed. This is not ideal since the information varies in each scorecard layout and is a mixture of handwriting and printed text. The solution from LEADTOOLS is not suited in terms of scope and implementation effort since it mainly supports other languages but Python. Furthermore, LEADTOOLS is too expensive to try out if its results are not sufficient. Finally, ABBYY was chosen since it fits the requirements of this thesis best and is free of charge for the first 500 pages. Zonal recognition is supported as well as a custom alphabet and rules. Besides that, ABBYY was recommended since it produces good results in similar fields of application.

### 5.5.2 ABBYY

To identify optimal settings of the ABBYY algorithm for the application, all features (full document and zonal recognition, recognizing single fields, and array of fields) are tested. Afterwards, the chosen setup is implemented and optimized. The following two functionalities of ABBYY are considered: The recognition of text in whole documents in contrast to the recognition of single parts. To decide whether the recognition works, eight self-written scorecards with different games and layouts are processed and it is attempted to improve the recognition through parameter tuning.

#### 5.5.2.1 Recognition of whole scorecards

In the first step these scorecards are processed with full-page recognition. The returned recognition shows that the computer font is recognized, but not the handwriting. Furthermore, the layout cannot be reconstructed from the output and therefore it cannot be decided which strings correspond to which texts in the scorecards.

To improve the output, only the handwritten table is processed. ABBYY's settings are defined to extract text from a table but fail to deliver reliable results. Therefore, the approach is changed to an individual part extraction called zonal recognition.

#### 5.5.2.2 Recognition of individual parts

Since the recognition of whole documents has not been successful, the processing of the individual move boxes is continued. To begin, certain parameters are classified as non-optimizable and set to their appropriate value. These parameters are “textType”, “writingStyle” and “markingType”, which are set to their final parameters as shown in Table 4 and later discussed. To optimize the parameters “letterSet” and “regExp” two approaches are evaluated. Either to adapt them iteratively with the course of the chess game or to define the regex and the letterset in a way that they can be applied to all moves in general.

In the iterative approach a virtual chess game is played parallel to the recognition and based on all possible moves in the current board state the recognition space is minimized. Initial testing shows that if a move is detected incorrectly, the board state for all subsequent moves is faulty and affects the parameters in a negative way. A user input correction could correct this error, but all subsequent moves would then have to be re-recognized by ABBYY. Thus, an iterative approach does not yield optimal results.

**Global definition of parameters**

Since the iterative approach failed to deliver the expected results, the general approach is pursued. To define the parameters in general, all possible move notations in the algebraic notation are considered. Of all optional abbreviations [15] the following are also considered:

- "x" = capture
- "+" = check
- "++" or "#" = checkmate

The parameter "letterSet" is set to all occurring characters in a chess PGN syntax. To create a correct regular expression that matches all move variations, the regex is evaluated with a list of valid and invalid moves, including the optional abbreviations mentioned above[15]. With the regex itself, only the syntax can be validated but not whether a move is possible or not. Based on these considerations, the values are defined as shown in the following table.

[15] Regex for PGN moves. Available: https://regex101.com/r/Qqh5Nf/8/tests [30.05.2020]

Table 4

Parameter definitions for the recognition of handwritten moves [16].

| Parameter | Description |
|---|---|
| textType | Defined as “handprinted”, as the moves are handwritten. |
| writingStyle | Set to “german”, as the evaluated scorecards are written exclusively by German writers. This refers especially to the numbers 1 or 7 which have a different writing style in other languages. |
| markingType | Set to "simpleText", after all the text is no longer inside of a box. |
| letterSet | Defined to all occurring characters in the SAN moves: “RNBQKOabcdefghx12345678=+#-” |
| regExp | Set to a self-implemented regex, which only accepts syntactically correct SAN moves: “(((\|([RNBQK](\|[a-h])(\|[1-8])(\|x)\|[a-h](\|[1-8])x))[a-h]([1-8]\|[18]=[RNBQ]))\|O-O(\|-O))(\|[+#]\|[+][+])” |

Once all parameters are defined the document is ready to be recognized. The functions “submitImage” [17] and “processFields” [18] are used to send a single image to ABBYY and process all fields in one request. To use the method “processFields” an XML file must be transmitted with the request body which contains the information of the coordinates of all boxes and the parameters mentioned in the table above. Based on the XSD schema file[16] a custom XML file creator is in place. An example of such a file with a custom request and response can be seen in the appendix 9.4.1 Input XML format.

Finally, an XML is received that contains the recognition of ABBYY. Each character has a confidence for its recognition. If there are several candidates for certain characters, they are sent along with the recognition confidence. An example output can be found in appendix 9.4.2 Output XML format.

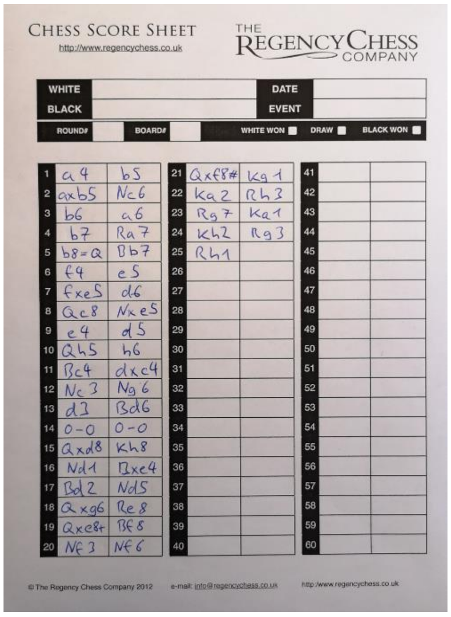
[16] ABBYY XSD Schema. Available: https://www.ocrsdk.com/schema/taskDescription-1.0.xsd [04.03.2020]

### 5.5.3 Results & Discussion

The approach with ABBYY is implemented and with it a robust recognition is achieved. It is possible to speed up the process by sending and receiving the information for a whole image/document instead of every move separately. The speed improves from two seconds per move to about three seconds for 40 moves. Overall a recognition confidence of around 95% is achieved. This meets the requirements of the application to present a solution to the problem definition. However, ABBYY is expensive in the long run and at the same time not optimized for the chess syntax. It is assumed that a better recognition can be achieved by an own CNN, therefore it is recommended to improve this approach in the future.

Furthermore, an evaluation is carried out to find common errors in ABBYY's recognition. These findings are further developed during this thesis.

To assess the common mistakes ABBYY yields during recognition, a recognition data set is created containing 21 scorecards (Figure 44). A scorecard of a match containing every character is selected and additional moves are appended to increase the number of rarely used characters. This scorecard is then transcribed by multiple writers. According to the circumstances the handwriting is more beautiful than at a tournament.



CHESS SCORE SHEET
http://www.regencychess.co.uk

THE REGENCY CHESS COMPANY

| | | | |
|---|---|---|---|
| WHITE | | DATE | |
| BLACK | | EVENT | |

ROUND# BOARD# WHITE WON ☐ DRAW ☐ BLACK WON ☐

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 1 | a4 | b5 | 21 | Qxf8# | Kg1 | 41 | | |
| 2 | axb5 | Nc6 | 22 | Ka2 | Rh3 | 42 | | |
| 3 | b6 | a6 | 23 | Rg7 | Ka1 | 43 | | |
| 4 | b7 | Ra7 | 24 | Kh2 | Rg3 | 44 | | |
| 5 | b8=Q | Bb7 | 25 | Rh1 | | 45 | | |
| 6 | f4 | e5 | 26 | | | 46 | | |
| 7 | fxe5 | d6 | 27 | | | 47 | | |
| 8 | Qc8 | Nxe5 | 28 | | | 48 | | |
| 9 | e4 | d5 | 29 | | | 49 | | |
| 10 | Qh5 | h6 | 30 | | | 50 | | |
| 11 | Bc4 | dxc4 | 31 | | | 51 | | |
| 12 | Nc3 | Ng6 | 32 | | | 52 | | |
| 13 | d3 | Bd6 | 33 | | | 53 | | |
| 14 | 0-0 | 0-0 | 34 | | | 54 | | |
| 15 | Qxd8 | Kh8 | 35 | | | 55 | | |
| 16 | Nd1 | Bxe4 | 36 | | | 56 | | |
| 17 | Bd2 | Nd5 | 37 | | | 57 | | |
| 18 | Qxg6 | Re8 | 38 | | | 58 | | |
| 19 | Qxe8+ | Bf8 | 39 | | | 59 | | |
| 20 | Nf3 | Nf6 | 40 | | | 60 | | |

© The Regency Chess Company 2012 e-mail: info@regencychess.co.uk http://www.regencychess.co.uk

Figure 44 Sample scorecard written by a volunteer to create an own recognition data set.

After the data collection phase, all moves are sent to the ABBYY. The test resulted in a matrix which is shown in appendix chapter 9.5.2. Of the various approaches, the recognition of the individual moves is implemented with generally applicable parameters.

The evaluation shows that a global recognition confidence of 95.46% is achieved. This number may vary when other scorecards or writers are evaluated and is calculated from 2644 total characters with 120 total recognition errors leading to this result.

This approach provides very accurate detection for well-legible moves. Based on the evaluation, it is possible to identify that there is a tendency for certain letters to be misrecognized.

The highest percentage error lies in character “a” with around 27% false positive recognitions. This is followed by “K”, “c”, “g” and “2” which all have around 13-16% false positive recognitions. A false positive recognition describes a case in which a character, i.e. “B” is recognized by ABBYY, but it is an “8” on the scorecard, leading to a false positive “B” recognition. Currently some “Q” and “g” are recognized as an “a” which are the previous 27% false positive recognitions. Since the writing area is very narrow, “Q” is often optically small and not clearly recognizable as “Q”, which means that they are recognized as “a”.

## 5.6 Improvement of recognition

After the whole game has been processed by the application, ABBYY's recognition is received. This output must be validated and improved since the output is not always correct. This chapter covers the steps taken to present the best possible guess for each move. To accomplish this the recognition data is post-processed and evaluated using a stochastic decision tree.

### 5.6.1 Post-processing of the ABBYY recognition

The recognition is first cleaned from all unwanted characters. For example, whitespaces are often among those. All characters are discarded that do not belong to the letterset as it was defined in Table 4. To revisit this definition, the letterset is defined as "RNBQKOabcdefghx12345678=+#-".

ABBYY does not return full words but combinations of single characters like: ("a", "x", "B", "5") which means "axB5" was recognized. If ABBYY is uncertain it returns options for each character in doubt: ("a", "x", ("B", "R"), "5"). Regarding the example, ABBYY is not sure if a "B" or an "R" is recognized and thus gives both a recognition confidence from 0 to 1.This is an example for a potential candidate for this index and these are extended in the following steps.

#### 5.6.1.1 Potential candidates per character

As discussed in chapter 5.5.3, the evaluation of ABBYY's recognition showed that certain characters tend to get classified incorrect. For example, it was found that only 73% of all the recognized "a" are true positives (Table 5). In this case all potential alternative characters which have an error rate >= 5% are considered as additional candidates.

Table 5

Extract from the ABBYY recognition evaluation showing the frequency (in %) of identified characters when "a" is the real value.

| Alphabet | a (%) |
|---|---|
| Q | 9.50 |
| B | 0.80 |
| d | 2.40 |
| a | 73.00 |
| g | 13.50 |
| 6 | 0.80 |

In the following example ABBYY recognizes character “a” with a recognition confidence of 100%. Subsequently the additional characters “g” and “Q” are added as possible candidates. These potential alternatives then reduce the recognition confidence of the letter “a” by the confidence of the candidates rounded up to 5% (Table 6). The final set of candidates and their confidences are stored until the next step.

Table 6

Potential candidate addition for the recognition of the character “a”.

| Recognition | | Candidates for “a” | | Adapted recognition | |
|---|---|---|---|---|---|
| Char: | Confidence: | Char: | Error rate: | Char: | Confidence: |
| “a” | 100.00% | “g” | 13.50% | “a” | 75.00% |
| | | “Q” | 9.50% | “g” | 15.00% |
| | | “d” | 2.40% | “Q” | 10.00% |

#### 5.6.1.2 Creation of move candidates

Now that the recognition has been cleaned and potential candidates were added, possible moves may be created based on the potential candidates of the characters. A move must be represented in a string form and is composed of a series of 2 to 8 characters. All possible combinations of the recognized characters and their potential candidates are generated, and their overall recognition confidence is calculated by multiplying their respective recognition confidences (Table 7).

Table 7

Combination of character candidates which lead to move candidates.

| First char | | Second char | | Move candidates | |
|---|---|---|---|---|---|
| Char: | Confidence: | Char: | Confidence: | Char: | Confidence: |
| “a” | 75.00% | “4” | 100.00% | “a4” | 75.00% |
| “g” | 15.00% | | | “g4” | 15.00% |
| “Q” | 10.00% | | | “Q4” | 10.00% |

#### 5.6.1.3 Creation of move containers

Once all potential move candidates have been determined, move container objects can be generated. Each move container corresponds to one move, initially containing the potential move candidates, which are processed in the decision tree. Furthermore, move containers are used to transfer the state of the application from backend to frontend and vice versa and to store all necessary information for later access (Table 8).

Table 8

A description of the move container object variables.

| Variable | Description |
|---|---|
| predicted_move | This string value is initially empty. If the tree algorithm successfully predicts a move, the value is set to the prediction. If the tree is stuck in a dead end, the value "Error" is set. |
| move_candidates | Includes all candidates that are created in the previous step. |
| move_certainty | Once the full decision tree has finished calculating a move, the move_certainty describes how certain the tree is, that this is the correct move. It contains a value from 0-1 where 1 is equal to 100%. |
| suggestions | The top predictions of the tree algorithm for this step if a move is successfully predicted. |
| status | Is set to a value that represents the status of this container:<br>• 0 = not processed<br>• 1 = validated by the user<br>• 2 = process error in the tree algorithm<br>• 3 = processed by the tree algorithm |

### 5.6.2 Stochastic decision tree

Each move container contains the move candidates and thus it is possible to estimate what was written in each box by the player. Since a chess game is correlated, it must be played from the first to the last move. Depending on the state of the board, only a finite number of moves are legal. Those moves are referenced as legal moves in the following subchapters. Incorrectly recognized chess moves can therefore change the state of the board in such a way that subsequent recognitions are no longer valid moves. To take advantage of this characteristic, a stochastic decision tree is implemented to consider subsequent moves in the prediction and ultimately improve the recognition. The following steps explain how the decision tree is implemented and how the algorithm operates.

#### 5.6.2.1 Structure of the tree

During a chess game, a new board state is formed after each move. These states are represented in the tree as nodes. The players' moves are represented as edges.

**Nodes**

A node contains all the information about the current board state which is displayed in Table 9. A node is expandable by adding an edge, connecting it to a new child node. The node value represents how promising this subtree is. The larger this is, the more likely are all the following nodes. To simulate the board state, the “python-chess” library is used [19]. It offers functions to validate individual moves given a certain board state. Furthermore, any given board state can be reproduced and validated.

Table 9

A description of the node object variables.

| Variable | Description |
|---|---|
| board | Contains the current board state as a board object created by the "python-chess" library. |
| value | Value of the node that represents how promising this subtree is based on its child nodes and is initially set to 1. |
| is_calculated | Is set to "True" if all legal moves have been evaluated and the likely edges have been created. Is used to avoid redundant calculations. |
| edges | A list of edge objects that connect this node with the following nodes. |

**Edges**

An edge represents a chess move and connects two nodes with each other. It includes the information of the heuristic certainty of the move and the two nodes which are connected (Table 10).

Table 10

A description of the edge object variables.

| Variable | Description |
|---|---|
| move | The string value of the chess move in SAN notation and which is a legal move in the current board state. |
| heuristic_certainty | The heuristic certainty that this legal move corresponds to the recognized move on the scorecard. This calculation is done with heuristics as described in the following chapter. |
| previous_node | The node from which this move originated. |
| following_node | The node that gets expanded when playing the current move in the board state of the previous node. |

#### 5.6.2.2 Expanding nodes with heuristics

All legal moves can be obtained from the board state. For each legal move an edge is created in which the move and the heuristic certainty value is stored. This value indicates how certain the algorithm is that the chosen legal move is the best possible guess for the recognized move. It is calculated with the distribution of heuristic points, which are described below. The previously mentioned move containers are used to access the move candidates and their information.

Each move candidate is compared with each legal move of the current board state. Based on their similarity, heuristic points are assigned to the legal moves. Because each move candidate also has his own recognition confidence, these are considered when adding up the points by multiplying the heuristic points (green) with the candidate's recognition confidence (orange) as shown in Table 11. A simplified example of this calculation with the same candidates from chapter 5.6.1.2 is displayed and only four legal moves are considered. The heuristic points presented here serve as an example and are fictional, to make it more comprehensible. As displayed in Table 11, candidates are compared to the currently legal moves. How the real heuristics perform is described after this calculation.

Strings that are more comparable get more heuristic points. Over all candidates the legal move "a4" has the most points since it is a full match and needs no character transformations to resemble a legal move.

Table 11

Fictional heuristic points distribution for legal moves from recognized move candidates.

| Legal moves / Candidates | a4 | g3 | h3 | Nf3 |
|---|---|---|---|---|
| a4 (75%) | 120 * 0.75 = **90** | 5 * 0.75 = **3.75** | 5 * 0.75 = **3.75** | 0 * 0.75 = **0** |
| g4 (15%) | 50 * 0.15 = **7.5** | 50 * 0.15 = **7.5** | 5 * 0.15 = **0.75** | 0 * 0.15 = **0** |
| Q4 (10%) | 5 * 0.10 = **0.5** | 5 * 0.10 = **0.5** | 5 * 0.10 = **0.5** | 0 * 0.10 = **0** |
| **Sum** | **98** | **11.75** | **5** | **0** |

The heuristic certainty of each legal move is then calculated as a percentage of the total sum of all legal moves. These percentages are output and represent the certainty that these moves correspond to the recognition.

$$a4 = \frac{98}{98 + 11.75 + 5 + 0} = 85.40\%$$
$$g3 = \frac{11.75}{98 + 11.75 + 5 + 0} = 10.24\%$$
$$h3 = \frac{5}{98 + 11.75 + 5 + 0} = 4.36\%$$
$$Nf3 = \frac{0}{98 + 11.75 + 5 + 0} = 0\%$$

The following section explains how the heuristic points are distributed (Table 12). To compare a legal move with a move candidate, fuzzy string matching is applied. Different algorithms are considered and those that turn out to be suitable for chess strings are chosen. The comparisons and their heuristic points are listed in the table below and the points were approximated with trial and error. A special case are strings with a length difference of two or more which are excluded from the heuristics and automatically receive zero points. Through comparing ABBYY's recognition with what was written on the scorecard it was found that every recognition has at most a difference of one character in length.

Table 12

Implemented heuristics to compare two strings.

| Name | Description | Possible Points |
|---|---|---|
| Full match | If the two strings are equal, return all possible points. | 5000 |
| Length | Return all possible points if both strings are equally long. Return 25 points less for each length difference. | 50 |
| Length substring | Search in both strings for the biggest matching substring. The longer the substring contrasts with the actual strings, the more points are returned. | 200 |

| | | |
|---|---|---|
| Levenshtein | The Levenshtein distance [20] is the minimum number of editing operations to get from one string to another. The higher this distance is, the fewer points are returned. | 100 |
| Matches from each side | In many cases, the positions of certain characters are correct when viewed either from the front or the back. Therefore, points are awarded per match up to a maximum of the possible points. | 500 |

#### 5.6.2.3 Discarding of unlikely edges

To make the decision tree more computable, a threshold is implemented which discards all edges with a smaller heuristic certainty than the threshold. To define a reasonable threshold range, random samples of the heuristic confidence distribution are taken based on the recognition data set used in chapter 5.5.3.

Two graphs of the heuristic certainty distribution of the same recognition in two different board states are shown as followed. The recognition contains the move candidates “Nc6”, “Ne6” and “Ng6”. In the left graph is a board state, which contains the legal move “Nc6”. Its certainty peaks over 60% due to a full match with a move candidate. This is different in the right graph, where no legal move has a full match, because the move candidates are not possible in the board state. However, the most similar legal moves peak slightly over 10%. Additional graphs are listed in the appendix 9.5.3.

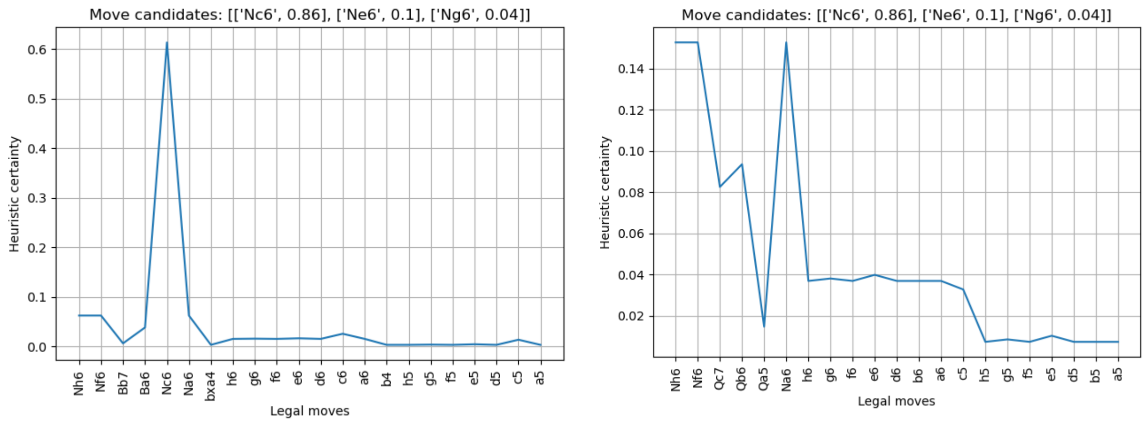


Figure 45 Different heuristic certainty distribution with move candidates “Nc6”, “Ne6” and “Ng6”, based on the legal moves of different board states.

The graphs show that uncertain values range between 0% and 5%, similar moves produce peaks in the range of 10% to 30% and matching moves peak over 50%. Based on these values, the threshold should be defined between 5% and 10% to always consider similar moves and not every possibility.

To locate the optimal percentage in this range, an automated test was performed on the recognition data set from chapter 5.4.3. For each setting, the number of errors over 19 scorecards is recorded. The depths range between 1 and 6 because manual tests have shown that higher depths can result in very long computational times.

The graph displayed in Figure 46 shows the cumulative error count per trial over all the scorecards.

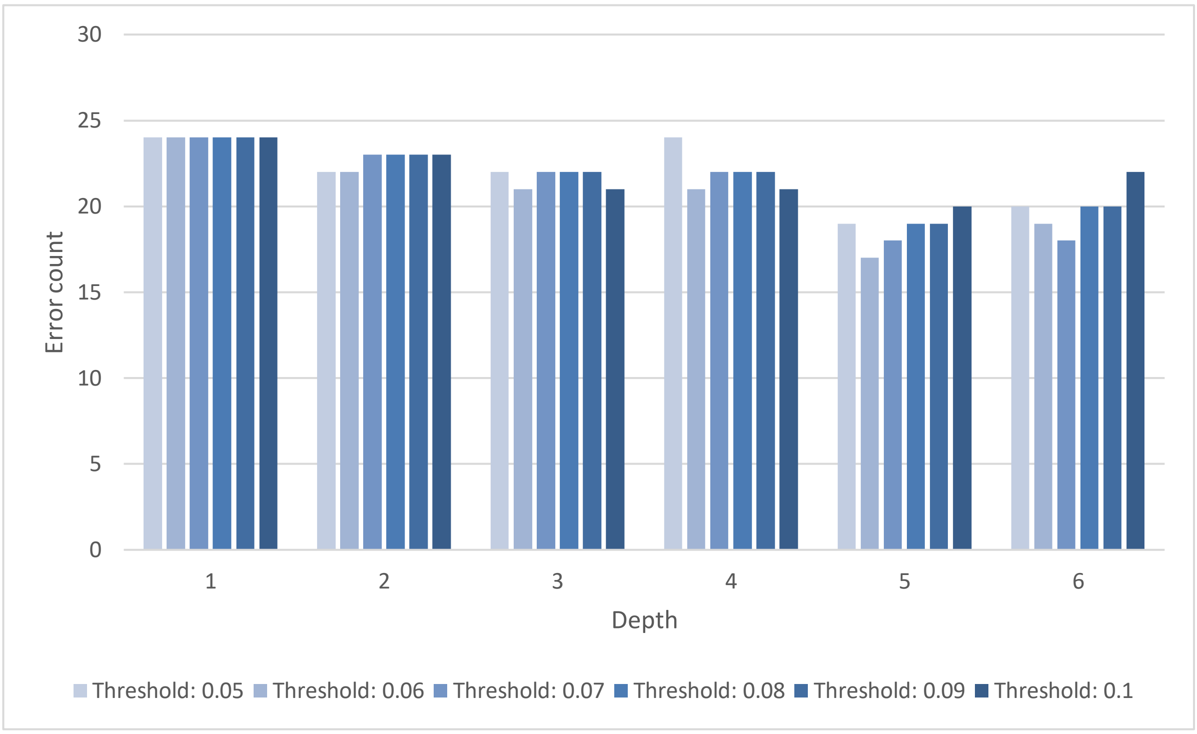


Figure 46 Error count per depth and threshold combination for 19 scorecards. Each scorecard contains a valid game of 20.5 moves.

The number of errors tend to decrease with increasing depth. However, with a depth of 6 the errors start to increase again. With increasing depth, faulty detections have a negative effect on more of the preceding predictions. However, with increasing depth more subsequent correct recognitions can positively affect the current prediction. It is assumed that in this case the drawbacks outweigh the benefits. Furthermore, if ABBYY returns an empty recognition, the depth can be specified however high, nevertheless the tree will not produce a reliable prediction.

To achieve the lowest possible error rate the most reliable configuration is with a depth of 5 and a threshold of 0.06. All further steps are implemented with these parameter settings.

To avoid dead ends, at least the three edges with the highest heuristic certainty are considered regardless of the defined threshold. It may be the case that several legal moves have the same heuristic certainty which can be seen in Figure 47. The move "a6" and the move "g6" are above the threshold and the moves "f6", "e6" and "d6" share the third highest heuristic certainty. Since it cannot be decided which of the edges is the most promising, up to 5 edges are considered in total. If more than 5 edges are chosen, the tree gets very broad too quickly and computation takes a long time.

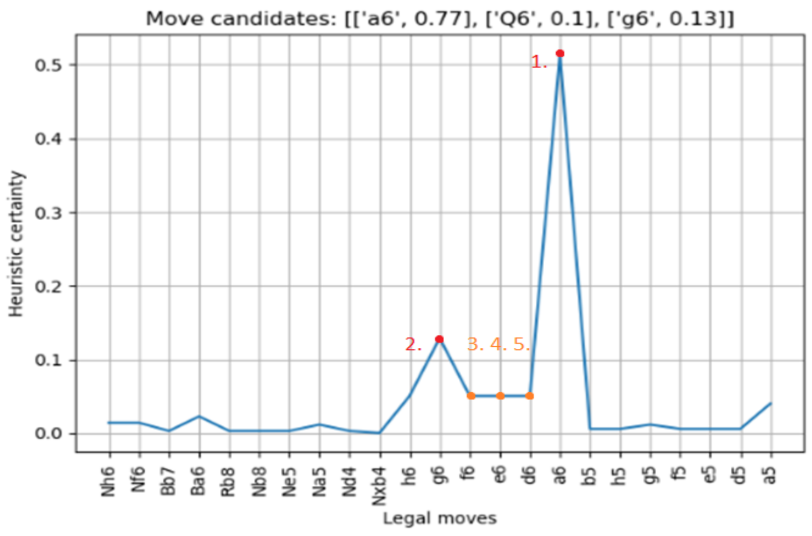


Figure 47 Example move candidates and the possible legal candidates in the board state. The numbers indicate all the selected moves.

A special case occurs if ABBYY returns an empty recognition. In this case it is not possible to tell which moves are possible and which are not. Therefore, none of the created edges are discarded and the heuristic certainties are overwritten with 100%. However, the disadvantage of this procedure is a very broad tree and a high computational time.

Considering the previous example from the heuristic point distribution, the discarding of unlikely branches is shown in the figure below. As can be seen, the heuristic certainty of the edges “h3” and “Nf3” are below the threshold value. But since the minimum amount must be met, the edge “h3” is not discarded. Therefore, only the edge “Nf3” is removed. Nodes are then created for all remaining edges with an initial node value of 1, in which the board state is extended by the chosen chess move.

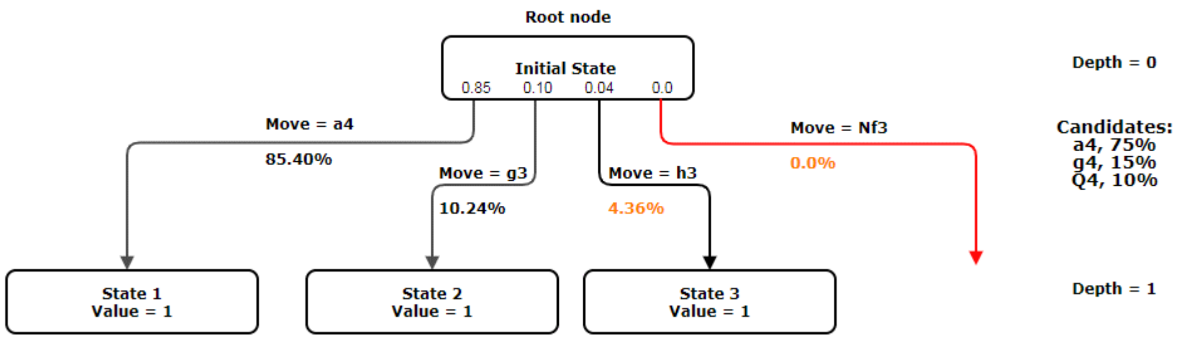


Figure 48 Decision tree with depth of 1. Orange heuristic certainties are below the threshold. Red edges are discarded.

### 5.6.2.4 Calculation of node values

Due to the creation of new nodes, the previous steps can be repeated until the specified depth is reached. First, all legal moves are taken into consideration. Then heuristic points and the certainty for all edges are calculated. Finally, the remaining edges are expanded. Once the specified depth is reached, the whole tree is created. For each node that exists at the maximum depth, a series of legal move combinations from the root node is found. These leaf nodes are initialized with a node value of 1. From the last layer the values are now calculated upwards. Each parent node multiplies the values of the child nodes with their edge heuristic certainty and creates a sum over them.

This calculation is shown in Figure 49, which is the expanded tree from Figure 48.

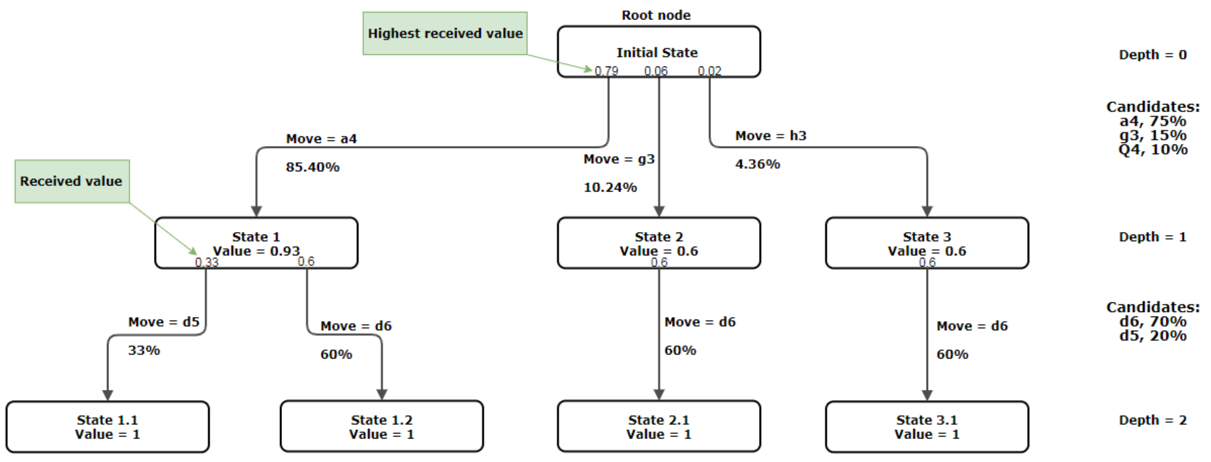


Figure 49 Decision tree with depth of two. The node values are recalculated and the edge of move “a4” is deemed the most promising.

#### 5.6.2.5 Determination of best guess

Once all node values have been calculated and propagated upwards, the root node receives the values from all its edges and selects the maximum. This value corresponds to the move that is the most promising from all initially considered legal moves in the root node. The chosen move is then written into the corresponding move container. In the example above, the move “a4” is chosen, because the value 0.79 is the highest received value. To calculate the move certainty of this max value in relation to all received values, the max value is divided by the sum of all values and then written into the move container:

$$move\ certainty = \frac{0.79}{0.79 + 0.06 + 0.02} = 0.91$$

The three most promising moves are added to the move container as suggestions for the user in the frontend interface. Finally, the corresponding child node to the chosen move is used as the new root node and the algorithm is recursively applied again.

#### 5.6.2.6 User input to correct moves

After all moves have been processed by the decision tree or if it has reached a dead end, the user is presented with the prediction. He is given the ability to change single predictions to correct eventual errors. As soon as the user corrects a move, the game is processed again by the algorithm. The validated and corrected moves are marked. The algorithm thus knows that at this node, only the marked move is possible and discards all other edges. Depending on the depth, this can affect previous moves as well as all subsequent moves.

### 5.6.3 Results & Discussion

The following evaluations are carried out with the aim of testing the individual steps and examining their improvement regarding the final prediction. For the following evaluations, the self-made data set from the ABBYY evaluation in chapter 5.5.3 is used. Out of the 21 scorecards 19 are used for this purpose since two contain invalid games and could not be tested for a full game recognition. The decision tree parameters are set to the defined threshold (0.06) and depth (5) from chapter 5.6.2.3 for the following evaluations.

#### 5.6.3.1 Improvement through additional candidates

To determine if the additional character candidates improve the output of the decision tree, a prediction is performed with and without the candidates.

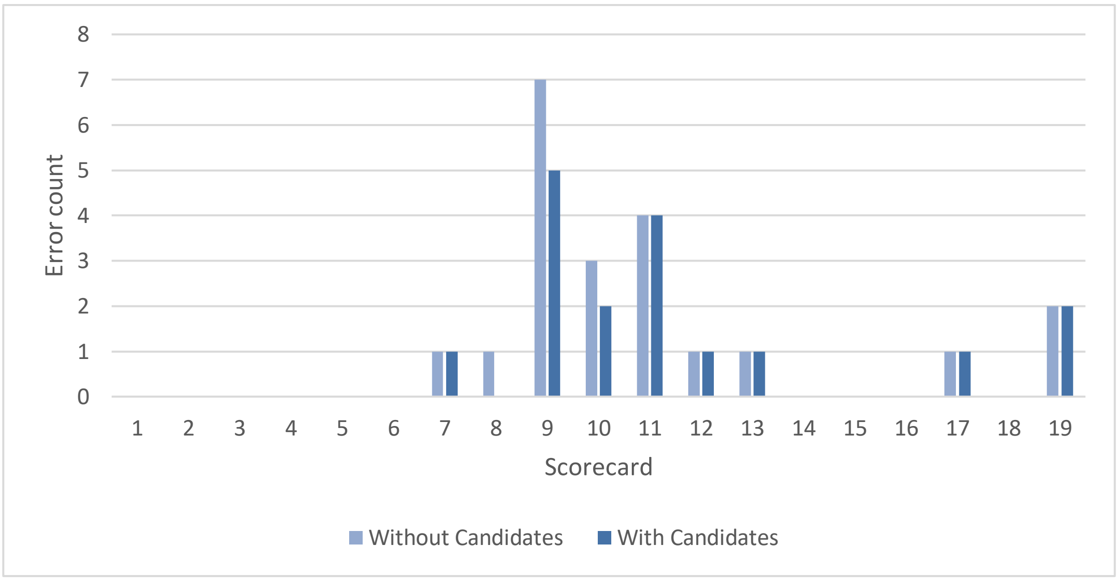


Figure 50 Error count of each scorecard with and without additional candidates. Each scorecard contains a valid game of 20.5 moves.

As Figure 50 shows, 10 out of 19 scorecards contained no errors. Scorecard 9 contains the most errors with and without additional candidates. The maximum number of errors is 7 which is recorded without additional candidates. The addition of candidates improves the prediction in 3 out of 9 scorecards and reduces the errors from a total of 21 to 17. This is an overall error reduction of 19% which indicates that approximately every fifth error could be resolved with this method. It also confirms the observed and chosen character transformations can be utilized like this.

#### 5.6.3.2 Evaluation of computation time

Once an edge is selected by the algorithm, most of the calculated information contained in the rest of the decision tree can be reused. Since the board states and their subsequent recognitions in the tree do not change, the calculated heuristic certainties are constant. Thus, they can be stored and only need to be recalculated in new layers. To confirm that the storage of data is saving time and improving performance, the computational time is evaluated with and without the storage of data over the data set.

Figure 51 shows the total error count of all 19 scorecards included in the data set. It shows the computational time in logarithmic scale for each scorecard.

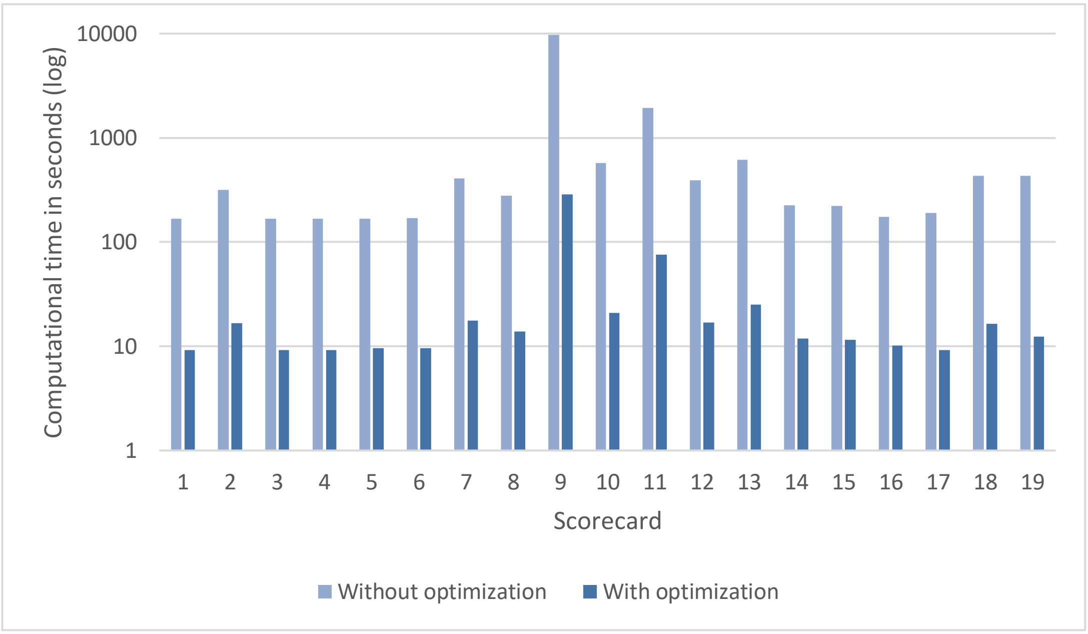


Figure 51 Computational time needed for each scorecard with and without optimization. Each scorecard contains a valid game of 20.5 moves.

Through storing already computed tree nodes, the computational time could be reduced by a factor of 18 to 30 depending on their amount of errors. Therefore, only the last depth must be recalculated. This is considered a success and paved the path for further utilization of the decision tree. Without this optimization, the decision tree would not be computable.

To measure the effect of different depths and thresholds on computability, the computation time per setting was measured including the optimized settings from the previous evaluation. To simulate a user interaction with the application, the game is first fully predicted before any errors are corrected. After the full prediction, the first error occurring in the prediction is corrected and the whole tree is recalculated. This procedure is repeated until all moves are correctly predicted.

Figure 52 shows the computational time in logarithmic scale versus the depth and threshold of the tree.

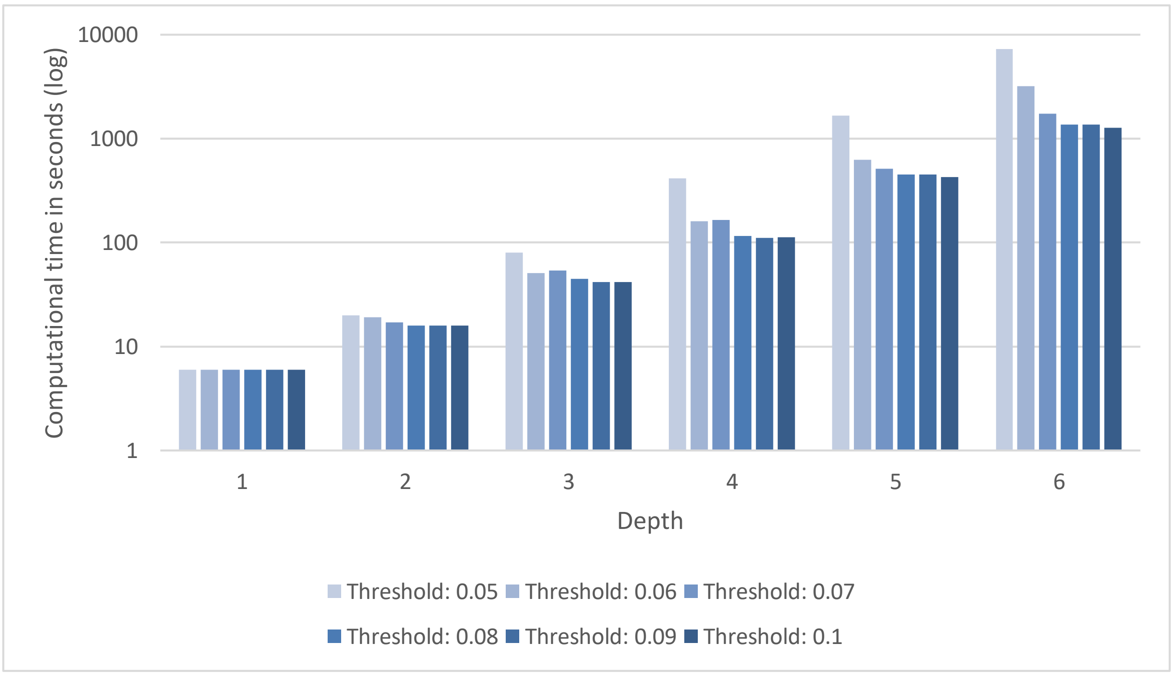


Figure 52 Computational time needed to predict and correct 19 scorecards per depth and threshold combination.

Figure 52 shows that the computation time per increased depth approximately triples. A threshold of 0.05 takes much longer to compute at higher depths. This is due to the width of the decision tree which is expanding rapidly as many moves have a confidence of over 5%.
The average computational time is 33 seconds per scorecard where each contain a game with 20.5 moves. Based on the database provided by "chessgames.com" [21], an average chess game is finished within 41.07 moves. Thus, the average computational time per game is around one minute for the user in the current setting.

The current implementation is not fully optimized and still suffers from long computational times. Multithreading or multiprocessing could be implemented to speed up the computation process.

#### 5.6.3.3 Performance of the overall implementation

The improvement of the chess move prediction using post-processing is investigated. Therefore, to compare the predictions, the output from ABBYY is compared with the output of the decision tree with the current settings.

The following figure shows how many errors per scorecard occurred through ABBYY's pure recognition and after post-processing.

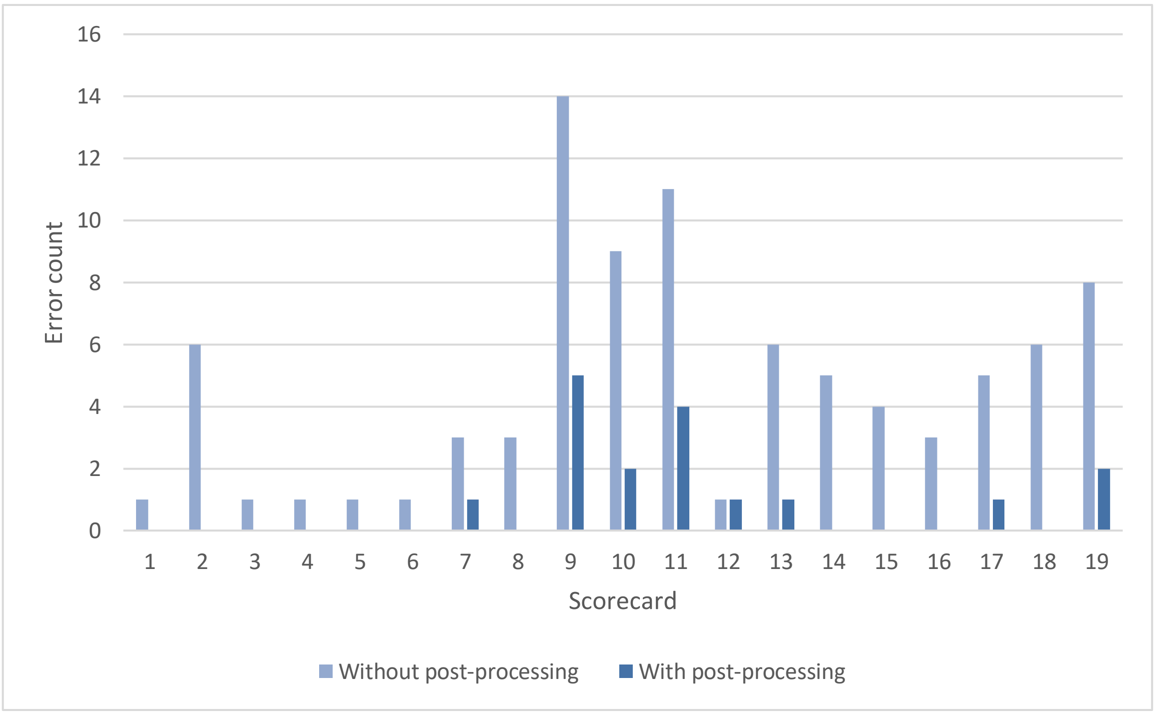
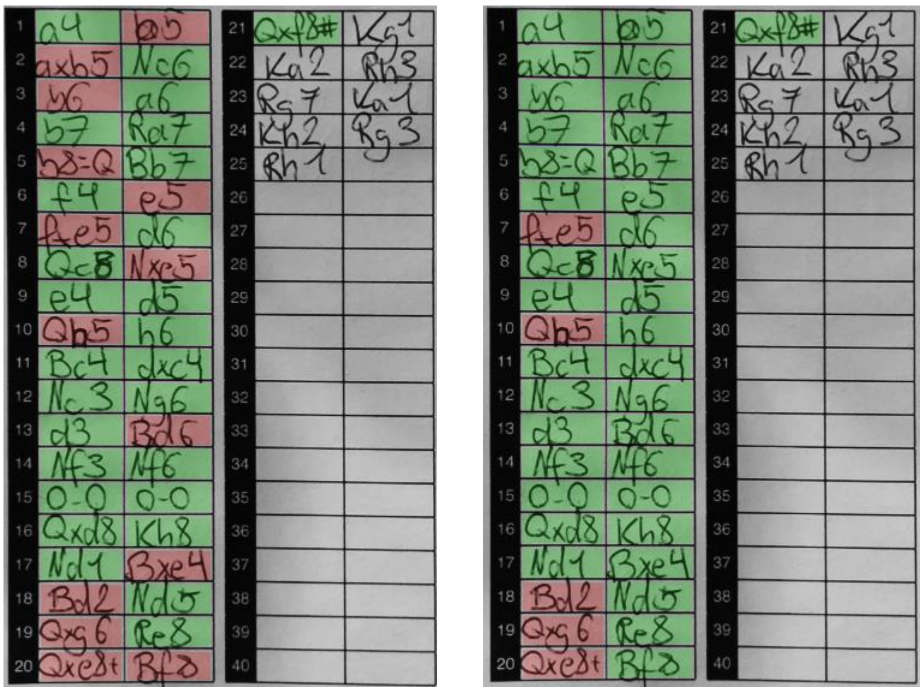


Figure 53 Comparison of errors per scorecard with and without post-processing. Each scorecard contains a valid game of 20.5 moves.

By validating on the data set, as well as post-processing and using heuristics in combination with a decision tree, the initial 89 errors were reduced to 17. This is an improvement of about 80% less errors. The tests have shown that minor errors can be corrected if they are followed by correct recognitions. Since moves in chess are interdependent, false recognitions are more likely to be corrected if they are in a relation or contradiction with the ensuing moves. Special cases with few errors exist, such as scorecard 7 or 12, where errors are so severe that they cannot be corrected automatically. To better understand the spike in errors recognized in scorecard 9, it is investigated in more detail below.

Figure 54 shows on the left side fourteen errors returned by ABBYY without post-processing. On the right side, it is reduced to five with the help of the decision tree algorithm.



| | | | | | |
|---|---|---|---|---|---|
| 1 | a4 | b5 | 21 | Qxf8# | Kg1 |
| 2 | axb5 | Nc6 | 22 | Ka2 | Rh3 |
| 3 | b6 | a6 | 23 | Rg7 | Ka1 |
| 4 | b7 | Ra7 | 24 | Kh2 | Rg3 |
| 5 | b8=Q | Bb7 | 25 | Rh1 | |
| 6 | f4 | e5 | 26 | | |
| 7 | fxe5 | d6 | 27 | | |
| 8 | Qc8 | Nxe5 | 28 | | |
| 9 | e4 | d5 | 29 | | |
| 10 | Qb5 | h6 | 30 | | |
| 11 | Bc4 | dxc4 | 31 | | |
| 12 | Nc3 | Ng6 | 32 | | |
| 13 | d3 | Bd6 | 33 | | |
| 14 | Nf3 | Nf6 | 34 | | |
| 15 | O-O | O-O | 35 | | |
| 16 | Qxd8 | Kh8 | 36 | | |
| 17 | Nd1 | Bxe4 | 37 | | |
| 18 | Bd2 | Nd5 | 38 | | |
| 19 | Qxg6 | Re8 | 39 | | |
| 20 | Qxe8+ | Bf8 | 40 | | |

Figure 54 The left image is the prediction for scorecard 9 without post-processing and the right image is with post-processing. Marked in green are the correct and in red the wrong predictions.

The prediction fails due to the wrong recognition supplied by ABBYY. A few examples are given:

- "Bd2" is an empty recognition
- "Qxg6" is recognized as "QK&"
- "Qxe8+" is recognized as "Qxd+"

This problem can only be tackled if the handwriting is nicer and does not overlap the table lines or other letters. Furthermore, another ICR engine could be used to improve the recognition output.

In conclusion, by using post-processing, heuristics and a decision tree model the errors on the data set are reduced. Without post-processing 89 errors exist over the whole recognition data set. This is reasonable since a confidence of 95.46% is measured for ABBYY's output which leads to around 113 wrongly classified characters in 2470 characters over the data set. This number is optimized to a final low of 17 errors, which is an error reduction by 80%. This shows that the decision tree gives a significant boost in the right direction.

There are many aspects which can be extended. If the heuristic points are to be better distributed, the current board state in each move should be considered. An idea was to implement a "MiniMax"[17] as a heuristic tool inside the decision tree implementation. This gives an insight into the current board state and helps to understand if a favourable position can be reached. When playing chess on an average to high level, the goal is to win and to outplay the opponent. Bad board states are not desired and will be avoided whenever possible. This could remove unnecessary branches and further reinforce legal moves.

By applying the minimum error configuration of a depth of 5 and a threshold value of 0.06, the lowest amount of errors is recorded in the data set. However, the computational time is considerably higher than using a lower depth. By reducing the depth, a slightly higher number of errors are predicted but it is faster for the user to correct more errors instead of waiting for the algorithm to calculate the decision tree on high depths. Furthermore, the decision to select a minimum of three and a maximum of five different legal moves could be rethought and optimized in additional steps in the future. Considering the time savings, the depth is set to 3 and the threshold to 0.06 in the productive system which overall enhances the user experience with lower idle times. Currently the recognized moves are sent back and forth in a JSON file format. The file size might exceed the maximum session file size of 4MB thus the end of a game might be cut of if it is longer than 40 moves. This could be fixed with user and session management to not always send the whole recognition but only the relevant changes back and forth.

[17] Game Theory – The Minimax Algorithm Explained. Available: https://towardsdatascience.com/how-a-chess-playing-computer-thinks-about-its-next-move-8f028bd0e7b1 [14.06.2020]

# 6 Conclusion

The goal of this bachelor thesis was to implement an algorithm which automatically digitizes the moves of a chess game from the image of a scorecard. This algorithm detects the handwritten moves, recognizes and checks them for correctness. The application was developed in Python with a Flask web server and contains a web interface for the user to interact with and partake in the digitization and recognition process.

The application takes an image of a scorecard with a table layout and outputs a PGN file containing the written chess game after taking in user input. The table must have solid lines and the initial handwritten text must be legible. Furthermore, the written text should be strictly written inside the boxes. The character recognition was implemented through the ABBYY API and a recognition confidence of around 95% was achieved. Additionally, based on commonly occurring recognition errors a stochastic decision tree with custom heuristics was implemented to improve the quality of the recognition and validate the moves. This reduced the error rate over the data set of 19 scorecards by around 80% from 89 down to 17 errors.

With these results, a reliable recognition was achieved. The software stands out from the competition since multiple layouts are supported and the application is interactive and control over the recognition is handed to the user.

# 7 Outlook

To further improve the application, the custom CNN approach could be investigated and implemented. A custom CNN might reach a higher confidence score than ABBYY and thus further improve the application. To support more layouts, the web interface could offer an option to pre-select the input scorecard layout and adapt the used recognition algorithm in the application regarding the selected option. The underlying algorithms may be improved to solely focus on the handwritten text instead of the table lines. Additionally, this approach could enable the extraction of header data including i.e. the tournament name, player names or the score of the game.

Since each step in the implementation offers potential for optimization, a documented test run of the application should be carried out to identify where the most errors occur. Once these points of failure are identified, they can be fixed successively.

To develop a commercially viable product, it requires a solid foundation and additional features:

- The frontend application should be ported to a current web framework.
- The architecture of the application should be adapted to function like an API to make it more extensible.
- The application in its current state only allows single user access and no user management system is in place. This extension would have to be installed to handle user accounts, user specific data and confidential information.
- Since the ABBYY OCR engine is not free of charge, an alternative and free approach would be beneficial to lower the cost of the application and allow for further specialization of the recognition.
- Further layouts could be supported within the application through focusing on the next most common layout apart from solid tables.
- Include chess analytics into the application and create an all-in-one solution to digitize and analyse chess games.

# 8 Listings

## 8.1 Bibliography

[1] Newsgroup rec.games.chess. Standard: Portable Game Notation Specification and Implementation Guide [Online].
Available:
http://www.saremba.de/chessgml/standards/pgn/pgn-complete.htm [01.04.2020]

[2] Newsgroup rec.games.chess. Standard: Portable Game Notation Specification and Implementation Guide [Online]. Sect. 8.2.3.
Available:
http://www.saremba.de/chessgml/standards/pgn/pgn-complete.htm [01.04.2020]

[3] S. Garrido-Jurado et al. 2014. Automatic generation and detection of highly reliable fiducial markers under occlusion. Pattern Recogn. 47, 6, June 2014, p. 2280-2292.

[4] G. Cohen, S. Afshar, J. Tapson and A. van Schaik, "EMNIST: an extension of MNIST to handwritten letters", The MARCS Institute for Brain, Behaviour and Development, Penrith, Australia, arXiv:1702.05373v2, 01 March 2017.

[5] A. Rosebrock. (2014 September 1). How to Build a Kick-Ass Mobile Document Scanner in just 5 minutes [Online].
Available:
https://www.pyimagesearch.com/2014/09/01/build-kick-ass-mobile-document-scanner-just-5-minutes/ [10.04.2020]

[6] CSE. (2018). Making sense of Handwritten Sections in Scanned Documents using the Azure ML Package for Computer Vision and Azure Cognitive Services [Online].
Available:
https://devblogs.microsoft.com/cse/2018/05/07/handwriting-detection-and-recognition-in-scanned-documents-using-azure-ml-package-computer-vision-azure-cognitive-services-ocr/ [10.05.2020]

[7] K. Vyas. (2018, July 22). A Box detection algorithm for any image containing boxes [Online].
Available:
https://medium.com/coinmonks/a-box-detection-algorithm-for-any-image-containing-boxes-756c15d7ed26 [07.04.2020]

[8] OpenCV. (2020). Morphological Transformations (4.3.0-dev) [Online].
Available:
https://docs.opencv.org/trunk/d9/d61/tutorial_py_morphological_ops.html [09.04.2020]

[9] J. Matas, O. Chum, M. Urban and T. Pajdla. (2004). Robust Wide Baseline Stereo from Maximally Stable Extremal Regions. Image and Vision Computing. 22. 761-767. 10.1016/j.imavis.2004.02.006.

[10] MathWorks. Automatically Detect and Recognize Text in Natural Images [Online].
Available:
https://www.mathworks.com/help/vision/examples/automatically-detect-and-recognize-text-in-natural-images.html [03.04.2020]

[11] H. Chen, S. Tsai, G. Schroth, D. Chen, R. Grzeszczuk and B. Girod. (2011). Robust text detection in natural images with edge-enhanced Maximally Stable Extremal Regions. IEEE International Conference on Image Processing. 2609-2612. 10.1109/ICIP.2011.6116200.

[12] ABBYY. Cloud OCR SDK [Online].
Available:
https://www.ocrsdk.com/ [20.04.2020]

[13] Leadtools. ICR SDK Technology [Online].
Available:
https://www.leadtools.com/sdk/ocr/icr [20.04.2020]

[14] Google Cloud. Detect handwriting in images [Online].
Available:
https://cloud.google.com/vision/docs/handwriting [20.04.2020]

[15] International Cheess Federation. (2018, January 1). Fide Laws of Chess [Online].
Appendix C.13.
Available:
https://old.fide.com/fide/handbook.html?id=208&view=article [28.04.2020]

[16] ABBYY. (2020) XML parameters of field recognition [Online].
Available:
https://www.ocrsdk.com/documentation/specifications/xml-scheme-field-settings/
[04.03.2020]

[17] ABBYY. (2020) API Reference submitImage Method (version 1) [Online].
Available:
https://www.ocrsdk.com/documentation/api-reference/submit-image-method/
[04.03.2020]

[18] ABBYY. (2020) API Reference processFields Method (version 1) [Online].
Available:
https://www.ocrsdk.com/documentation/api-reference/process-fields-method/
[04.03.2020]

[19] N. Fiekas. (2020). Python-chess, Project description [Online].
Available:
https://pypi.org/project/python-chess/ [16.04.2020]

[20] Cluelogic Insights. (2017 January 25). The Levenshtein Algorithm [Online].
Available:
https://www.cuelogic.com/blog/the-levenshtein-algorithm [01.06.2020]

[21] Chessgames Services LLS (2020). Statistics Page [Online].
Available:
https://www.chessgames.com/chessstats.html [10.06.2020]

## 8.2 List of Figures

## 8.3 List of Tables

# 9 Appendix

## 9.1 Official task, project assignment

**Bachelor thesis definition**

| | |
|---|---|
| Topic: | OCR for Chess Scorecards [Machine Learning] |
| Students: | Colin Dreher (drehecol), Béla Noah Horváth (horvabel) |
| Supervisor: | Prof. Dr. Mark Cieliebak |

Problem definition:
Nowadays chess players write down the moves of a chess game in a Portable Game Notation (PGN) on a spreadsheet while they are playing. If the players want to analyse their game, they have to replay all the moves by hand in a program such as "ChessBoard". This takes a lot of time and is sometimes not worth the effort for the players, although such analysis programs can give the players valuable information about their behavior and tactics.

Objective:
The aim of this bachelor thesis is to implement an algorithm, which processes an image of a scorecard. The algorithm should recognize the players moves with machine learning, check them for correctness and return them as a machine readable PGN.

The algorithm should be implemented in Python on a web server. The webapp should have an interface for the user to interact with the PGN output to check the validity of a PGN output or redefine uncertain or incorrect moves.

## 9.2 Software architecture

### 9.2.1 Physical architecture

The software is divided into two parts, the backend, and the frontend. The backend includes all tasks for the image processing, while the frontend is used for presentation and interaction. The frontend depends on user input to either confirm or correct the presented recognition, computed in the backend.

For the backend, a Python application is used. To allow the backend to communicate with the frontend the framework Flask[18] is chosen. Flask is an open-source web framework that aims to have a small core while remaining extensible. Flask only depends on Jinja2[19] as a templating engine and Werkzeug[20] as a WSGI interface. Flask has its own web server for development and debug purposes which is suitable for this work. In this thesis, the ABBYY application programming interface (API) is used for all OCR tasks.

The frontend is developed without a framework and uses the templating engine of Flask to produce a dynamic single page application.

To deploy the application, it is recommended to use a different web server with a WSGI. The application is hosted on a virtual server from Cloudlab at the ZHAW[21]. To deploy the application on said server, the mod_wsgi package[22] is used.

[18] Pallets Flask documentation. Available: https://flask.palletsprojects.com/en/1.1.x/ [18.05.2020]
[19] Pallets Jinja documentation. Available: https://palletsprojects.com/p/jinja/ [18.05.2020]
[20] Pallets Werkzeug documentation. Available: https://palletsprojects.com/p/werkzeug/ [18.05.2020]
[21] InIT Openstack Clusters. Available: https://info.cloudlab.zhaw.ch/pages/openstack.html#cluster-apu-apu-cloudlab-zhaw-ch [18.05.2020]
[22] Documentation mod_wsgi. Available: https://modwsgi.readthedocs.io/en/develop/ [04.05.2020]

The image below shows the situation where the application is deployed and a user accesses it via his browser.

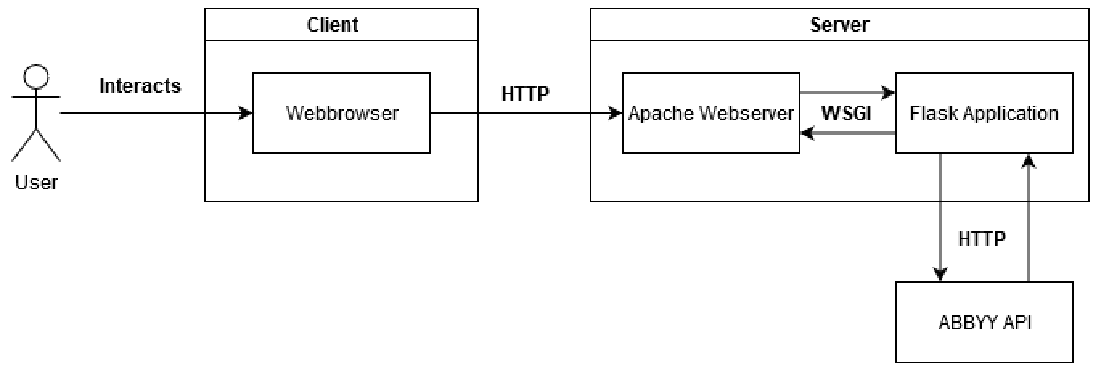


Figure A - 1 Physical architecture of the application.

### 9.2.2 Logical architecture

As stated, a backend and a frontend structure is used. The following chapter describes the flow of the application in more detail. Additionally, all contained files are listed and their purpose is documented.

#### 9.2.2.1 Backend

The backend is a Python application. Python is suitable for machine learning and image processing. Since it must be possible to handle all subtasks at the same place and most of the existing solutions are written in Python, Python is a solid choice.

Every folder must contain a "__init__.py" file to ensure that the compiler has access to all the modules in the project. When deploying the server with apache and mod_wsgi, as described in the appendix 9.3.2, the flask app must be created inside an "__init__.py" file, for it to correctly instantiate and run.

The backend is then divided into five main parts. These parts are, Flask, Scanner, Pre-processing, ABBYY and Post-processing. An additional directory called Helper contains functionality used over the whole application and it is not considered as its own part. The last directory is called Evaluations and contains a test environment to probe heuristics and decision tree parameters. In each of these, the available scripts and their scope of functions are described.

If an error occurs in any of the steps below, a custom exception is thrown and the user is redirected onto the error page with a custom message. This helps to understand what went wrong and how to tackle the problem.

**Implementation**

The central entry point is the file "__init__.py" in the top directory. This file contains the flask web server and communicates with the frontend. To call the algorithms from within the flask server and apply them to the uploaded images, another central file is needed. This file is called "frontendPipeline". All partial steps can be called from within this file.

Table A - 1

Central backend files of the application.

| File name | Description |
|---|---|
| __init__ | Central access point of the application and container of the flask web server. |
| frontendPipeline | The frontend pipeline contains all necessary methods to run the application from start to finish and is the central interface between the flask web server and the algorithm. |

**Helper**

To ensure that certain conventions are adhered throughout the application and functionality is not reimplemented several times, "helper classes" are used and they can be seen in Table A - 2. These files are imported when they are needed and serve to ensure a more uniform structure.

Table A - 2

All files that contain classes or helper functionality.

| File name | Description |
|---|---|
| exceptions | List of custom exception classes. |
| jsonParser | Helper methods that parses the JSON into move container objects or move container objects into JSON. |
| enumClasses | Global values sorted into different Enums. Used within the application. |

| | |
|---|---|
| moveContainerClass | Contains the moveContainerClass with its features. Used to create move container objects and store information inside them. Provides the functionality to dump the object as JSON and to interact with the move itself. Add suggestions, recognitions or change the status of the move. |

**Scanner**

If the application is entered via the "frontendPipeline" the uploaded image is aligned first. This is achieved with two files listed in Table A - 3. The “transformer” file contains functionality to mathematically transform the image in a plane.

Table A - 3

All files containing the logic to align images.

| File name | Description |
|---|---|
| scanner | The scanner module which aligns the uploaded images. |
| transformer | An additional module of the scanner which is responsible for the "4-point" transformation. Four points, so all vertices of the scorecard are needed to perform the transformation. |

**Pre-processing**

After the image is aligned it needs to be pre-processed first. Pre-processing is split into multiple parts, as can be seen in Table A - 4. First “preProcessImage” is called and creates a custom directory for each upload. This directory is then filled with the pre-processed images in binary format.

After all the image pre-processing has been carried out, “splitTableInBoxes” is called. As the name suggests, all fields in which the player can write are searched in the image and thus the table is split into boxes. Some layouts may have row numbering inside these boxes. With the help of "leadingNumerRemover" this enumeration is removed. Furthermore, the user is asked to validate the found boxes via unselecting all irrelevant boxes and continuing as described in chapter 4. When continuing, “removeUnusedBoxes” is called to remove those that are unselected and so only the boxes containing handwritten moves are left. After this step, all box coordinates and the width and height of all the boxes is calculated and the recognition can be performed.

To prepare for a custom CNN recognition, all characters must be separated from each other since a CNN classifier on EMNIST data operates on single characters instead of whole words. The files “findCharactersInBox” and “preProcessEmnist” implement this but are not used in the productive system.

Table A - 4

All files containing pre-processing functionality.

| File name | Description |
|---|---|
| preProcessImage | All the pre-processing is handled in this file. It contains methods for removing the shadows and applying the kernel. |
| splitTableInBoxes | Extracts all relevant boxes from the binary image. It is the next step after the image has been pre-processed. In this file also the "leadingNumberRemover" is executed to remove parts of the boxes if necessary. |
| leadingNumberRemover | This file contains the utilities to recognize a number pattern which enumerates the lines of the scorecard. If a pattern is found, it will iteratively remove the pattern to ensure that the removed part is not too wide. |
| removeUnusedBoxes | Receives all boxes and all the relevant boxes. Compares both lists and removes any that are inside the relevant boxes. This is due to the interaction with the frontend where the user unselects any boxes that contain no handwritten move. |
| findCharactersInBox | After finding all boxes in the image, those boxes are further split into all their contained characters. |
| preProcessEmnist | Pre-processes the found characters inside “findCharactersInBox” to match the EMNIST format. This is necessary if a CNN is trained with EMNIST data since the same format must be present. |

**ABBYY**

Once all images are pre-processed and merely the relevant boxes are left, recognition can be performed with ABBYY. The settings, as well as creating a task and addressing the API, are regulated in the two files in Table A - 5. To recognize certain regions in a whole document at once, also called zonal recognition, an XML document must be passed to the API. This document gets created inside "AbbyyOnlineSDK".

Table A - 5

All files that are needed to use the ABBYY.

| File name | Description |
|---|---|
| AbbyyOnlineSDK | Contains the AbbyyOnlineSdk class according to the template of ABBYY. Additionally, contains helper methods to create and parse the XML file back and forth. Its main purpose is to keep track of the current task created in the "process". |
| process | Contains the setup of ABBYY, creates the processor and includes the actual method "recognize_image" which is used for recognition. It is in close cooperation with AbbyyOnlineSDK. |

**Post-processing**

After receiving the data from ABBYY, the important fields of the returned XML document are parsed and cleaned up using "post-processing". The cleaned data is the basis of the "decisionTree" which uses the "stringHeuristics" to create an optimized chess game. Finally the chess game is sent to the frontend as JSON file and displayed to the user. This JSON file sometimes exceeds the maximum session size of 4MB and thus not all the information can be sent to the frontend.

Whenever the user changes a move, it gets marked to "editedByUser" and thus is seen as correct. The new, changed chess game is now returned to the backend and again processed by the "decisionTree". When finished, the user gets a revalidated version of his game which is again interactable until the user decides to download his final PGN file as an output.

Table A - 6

All files containing functionality that is applied after the recognition of ABBYY.

| File name | Description |
|---|---|
| post-processing | "post-processing" post-processes the recognition of ABBYY. This is where the recognition is cleaned up for spaces and noise. Additional candidates are also added based on the common recognition flaws. Based on these character candidates, move candidates are created which are stored in a move container object. |
| decisionTree | The decisionTree file includes an implemented stochastic decision tree. This tree is applied to the data to find the most optimal chess game, which is later transferred to the frontend. |
| stringHeuristics | Includes all heuristic methods that are used to compare two strings on how similar they are. These methods are used in the decision tree to compare legal moves with the post-processed recognition. |

**Evaluations**

This is not part of the productive system and only serves evaluation purposes. It is used to experiment with different heuristics and decision tree parameters.

Table A - 7

All files containing evaluation functionality.

| File name | Description |
|---|---|
| ABBYYoutputEval | This evaluation counts the amount of wrong recognized moves by ABBYY. Outputs the errors and counts in a text file. |
| depthThresholdTreeEval | Contains the logic to test the decisionTree with different thresholds and depths. It counts the occurred errors and measures the process time with each depth and threshold. Outputs a text file that contains a summary of the test. |
| detailedTreeEval | Evaluates all scorecards with a defined threshold and depth overall scorecards. Collects information about the errors for each scorecard. Outputs a text file with all this information. |

#### 9.2.2.2 Frontend

The frontend is implemented without a framework. This presents the freedom to change the design and the process as it was considered best practice per task. However, a framework such as Vue, React or Angular can be recommended if the complexity of the application increases.

Currently the Jinja2 templating engine provided by Flask is used to dynamically load and populate the templates. The description of the frontend is reduced to a list of files and the folder structure which can be seen in appendix 9.3.3.

**Templates**

The templates directory contains all the views of the frontend.

Table A - 8

All templates of the application in the frontend.

| File name | Description |
|---|---|
| base | Base file with containers. Gets filled dynamically with content from the following files. |
| landingPage | Starting page that gets displayed when starting the web server. Contains the Upload functionality. |
| confirmBoxes | Contains the stepper with the box confirmations and info boxes. The logic behind the green/blue buttons is handled in the confirmBoxes JavaScript file. |
| downloadPage | Handles the download of the actual PGN file as well as the interaction with the output. Allows users to reset the table if any mistakes were made and offers the feature to revalidate the PGN output by the application. |
| error | Displays custom error messages and gets loaded inside base. |

**Partials**

Every template is considered a partial, if it does not represent a whole page but merely a reusable fragment.

Table A - 9

All files that are implemented as partials. Can be included in every page.

| File name | Description |
|---|---|
| loadingScreen | Contains the loading screen that is used as a partial and included in the html sites that need a loading screen for async calls. |
| tournamentForm | Is a modal in itself and gets called when trying to download the final PGN file. Contains the form to add the event, site, players and result of the game. |
| infoBox | The container of the info box with its overlay. Can be dynamically filled with content through JavaScript and inserted withing the application in each step. |

**Static**

The static directory contains all images, stylesheets and JavaScript files. They must reside in this folder to be served by flask, depending on the current request of the browser.

**Images**

Table A - 10

All the image directories and their corresponding task.

| File name | Description |
|---|---|
| favicon | Contains the sites favicon. |
| icons | Contains logos and interactive pictures. |
| processedUploads | Contains the processed uploads by their filename after the application processed an upload. Also contains info box images. |
| uploads | Folder in which all uploads from “landingPage” are stored if they are successfully uploaded. |

**Javascript**

Table A - 11

All JavaScript files and what they are used for.

| File name | Description |
|---|---|
| upload | Logic to upload an image to the backend flask server. |
| confirmBoxes | Contains the logic to validate the found boxes in the scorecard. Uses a state to display the red/green or grey/blue selection process. Also sends the final layout to the backend to finally recognize the moves with ABBYY Cloud SDK OCR. |
| downloadPage | Used to interact with the recognition, edit and revalidate the recognition. To revalidate it sends the table in a JSON format back to the backend and receives a new table in a JSON format with a revalidated game. Also contains the logic to navigate the table and ultimately download the PGN file. For usability purpose a reset button is present to reset the table to its initial state. |
| jquery | jQuery library used for DOM interaction manipulation. |

**Styles**

Table A - 12

All stylesheets and their scope of usage.

| File name | Description |
|---|---|
| style | Global stylesheet. |
| post-processing | Styling of the post-processing that contains the stepper, buttons and confirmBoxes html page. |
| loadingScreen | Styling for the partial page loadingScreen. |
| turnamentForm | Styling for the partial page tournamentForm. |
| infoBox | Styling for the partial page infoBox. |

## 9.3 Technical documentation

This part describes the full installation process to locally develop or to deploy the app on a server.

### 9.3.1 Local installation

**Repository data:**

Fork the repository: https://github.zhaw.ch/horvabel/Bachelorarbeit

Clone the repository locally or copy the files into a folder and create a new repository.

**ABBYY Setup:**

Create a new account: https://www.ocrsdk.com/

Check your E-mail for the Application Password. Copy the password.

from: https://cloud.ocrsdk.com/Account/Welcome

copy your Application ID in a similar format "551ff3ed-40d8-4f0f-8d2b-7150d25861de"

Open AbbyyOnlineSdk.py file located in "./Algorithm/ABBYY_OCR/AbbyyOnlineSdk.py" and change the ApplicationId and Password accordingly inside AbyyOnlineSdk class.

This allows you to use the free 500 pages. If a licence is in place you must use the account details of the licenced account.

**Backend Setup:**

1. Install Python version 3 (sudo apt-get install python3.X)
2. Install pip3 python3-pip (sudo apt-get install pip3)
3. Install python virtualenv: pip install -upgrade virtualenv
4. Create and activate the venv in the app folder

***For Windows (Python v. 3.8):***

*cd "appname"*

*virtualenv -python "c:\python38\python.exe" env .\env\Scripts\activate*

***PyCharm:***

*Create virtualenv in interpreter settings.*

*Activate venv for the project*

5. Install requirements:

   *pip3 install -r requirements.txt*

### 9.3.2 Deployment

Follow:

- https://apu.cloudlab.zhaw.ch
- https://www.youtube.com/watch?v=YFBRVJPhDGY
- https://stackoverflow.com/questions/39418012/my-apache-wsgi-flask-web-app-cannot-import-its-internal-python-module

to have as a backup when configuring the mod_wsgi. You can also choose another web server and gateway of choice like Nginx or similar.

**<u>Installation</u>**

1. Set up server like documented on cloudlab.
   - Forward port 80 (or 443 if needed) to the server.
2. Install Python and Pip (sudo apt-get install python)
   - sudo apt update
   - sudo apt install software-properties-common
   - sudo add-apt-repository ppa:deadsnakes/ppa
   - sudo apt install pythonX.Y (<-- enter version)
   - sudo apt install python3-pip
3. run: sudo pip3 install -r requirements.txt (located in base of repository)

   **UBUNTU ONLY**
   - . sudo apt install tesseract-ocr (use this if it cannot run because of tesseract ocr error)

4. Follow: https://www.youtube.com/watch?v=YFBRVJPhDGY tutorial on how to deploy with apache wsgi.
   - NOTE: https://stackoverflow.com/questions/39418012/my-apache-wsgi-flask-web-app-cannot-import-its-internal-python-module answer to configure the YOURAPPNAME.wsgi file, otherwise it will not work correctly.
5. Connect to the server via the public IP-Address and use your page!

### 9.3.3 Overview of the files

The following tree structure shows the whole application:

```
Project Code/
├── __init__.py
├── Algorithm/
│   ├── __init__.py
│   ├── ABBYY_OCR/
│   │   ├── __init__.py
│   │   ├── AbbyyOnlineSdk.py
│   │   └── process.py
│   ├── frontendPipeline.py
│   ├── Helper/
│   │   ├── __init__.py
│   │   ├── enumClasses.py
│   │   ├── exceptions.py
│   │   ├── jsonParser.py
│   │   └── moveContainerClass.py
│   ├── PostProcessing/
│   │   ├── __init__.py
│   │   ├── DecisionTree/
│   │   │   ├── __init__.py
│   │   │   ├── decisionTree.py
│   │   │   └── stringHeuristics.py
│   │   └── postProcessing.py
│   ├── PreProcessing/
│   │   ├── __init__.py
│   │   ├── findCharactersInBox.py
│   │   ├── leadingNumberRemover.py
│   │   ├── preProcessEmnist.py
│   │   ├── preProcessImage.py
│   │   ├── removeUnusedBoxes.py
│   │   └── splitTableInBoxes.py
│   └── Scanner/
│       ├── __init__.py
│       ├── scanner.py
│       └── transform.py
├── evaluations/
│   ├── ABBYYoutputEval.py
│   ├── depthThresholdTreeEval.py
│   └── detailedTreeEval.py
├── README.md
├── requirements.txt
├── static/
│   ├── images/
│   │   ├── favicon/
│   │   │   └── favicon.ico
│   │   ├── icons/
│   │   │   ├── anleitung.png
│   │   │   ├── arrow.png
│   │   │   ├── logo.png
│   │   │   └── newlogo.png
│   │   ├── processedUploads/
│   │   │   ├── aligned_template.PNG
│   │   │   ├── final_output.gif
│   │   │   ├── select_last.gif
│   │   │   ├── templatemove.png
│   │   │   └── unselect_boxes.gif
│   │   └── uploads/
│   ├── js/
│   │   ├── confirmBoxes.js
│   │   ├── downloadPage.js
│   │   ├── jquery.js
│   │   └── upload.js
│   └── styles/
│       ├── partials/
│       │   ├── infoBox.css
│       │   ├── loadingScreen.css
│       │   └── tournamentForm.css
│       ├── postProcessing.css
│       └── style.css
└── templates/
    ├── base.html
    ├── confirmBoxes.html
    ├── downloadPage.html
    ├── error.html
    ├── landingPage.html
    └── partials/
        ├── infoBox.html
        ├── loadingscreen.html
        └── tournamentForm.html
```

## 9.4 XML

Following, the XML input and output format for the ABBYY API function “processFields” are presented. The input format contains the settings for the recognition and the location of each move in the image. The image must first be uploaded with the “submitImage” function.

The output XML contains for each moveField a recognition and additionally “charRecVariants” if any exist. Meaning if an “e” gets recognized as char, it may contain a “c” as a charRecVariants, since e and c often get misclassified in handwriting recognition.

### 9.4.1 Input XML format

```
<document xmlns="http://ocrsdk.com/schema/taskDescription-1.0.xsd"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:tns="http://ocrsdk.com/schema/taskDescription-1.0.xsd">
      <fieldTemplates>
            <text id="moveField" top="0" right="0" bottom="0" left="0">
                  <language>English</language>
                  <textType>handprinted</textType>
                  <regExp>((((|([RNBQK](|[a-h])(|[1-8])(|x)|[a-h](|[1-
8])x))[a-h]([1-8]|[18]=[RNBQ]))|O-O(|-O))(|[+#])</regExp>
                  <markingType>simpleText</markingType>
                  <letterSet>RNBQKOabcdefghx12345678=+#-</letterSet>
                  <writingStyle>german</writingStyle>
            </text>
      </fieldTemplates>
      <page applyTo="0">
            <text template="moveField" id="0" top="665" right="613" bot-
tom="698" left="358"/>
            <text template="moveField" id="1" top="667" right="915" bot-
tom="702" left="625"/>
            <text template="moveField" id="2" top="710" right="613" bot-
tom="743" left="358"/>
            <text template="moveField" id="3" top="712" right="915" bot-
tom="746" left="625"/>
            <text template="moveField" id="4" top="754" right="613" bot-
tom="788" left="358"/>
            <text template="moveField" id="5" top="757" right="915" bot-
tom="791" left="625"/>
      </page>
</document>
```

Figure A - 2 The XML format used in the application to communicate with the ABBYY API function "processFields".

### 9.4.2 Output XML format

```
<?xml version="1.0" encoding="utf-8" standalone="yes"?>
<document xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://ocrsdk.com/schema/resultDescription-1.0.xsd
http://ocrsdk.com/schema/resultDescription-1.0.xsd"
xmlns="http://ocrsdk.com/schema/resultDescription-1.0.xsd">
      <page index="0">
            <text id="0" left="358" top="665" right="613" bottom="698">
                  <value>e4</value>
                  <line left="402" top="669" right="451" bottom="697">
                        <char left="402" top="674" right="420"
bottom="696">e</char>
                        <char left="431" top="669" right="451"
bottom="697">4</char>
                  </line>
            </text>
            <text id="1" left="625" top="667" right="915" bottom="702">
                  <value>e6</value>
                  <line left="714" top="673" right="760" bottom="701">
                        <char left="714" top="682" right="738"
bottom="701">e</char>
                        <char left="742" top="673" right="760"
bottom="701">6</char>
                  </line>
            </text>
            <text id="2" left="358" top="710" right="613" bottom="743">
                  <value>d4</value>
                  <line left="402" top="711" right="450" bottom="742">
                        <char left="402" top="711" right="425"
bottom="742">d</char>
                        <char left="432" top="714" right="450"
bottom="742">4</char>
                  </line>
            </text>
            <text id="3" left="625" top="712" right="915" bottom="746">
                  <value>d5</value>
                  <line left="692" top="712" right="745" bottom="745">
                        <char left="692" top="712" right="714"
bottom="745">d</char>
                        <char left="721" top="715" right="745"
bottom="745">5</char>
                  </line>
            </text>
            <text id="4" left="358" top="754" right="613" bottom="788">
                  <value>Nd2</value>
                  <line left="405" top="754" right="482" bottom="787">
                        <char left="405" top="760" right="425"
bottom="787">N</char>
                        <char left="423" top="754" right="446"
bottom="787">d</char>
                        <char left="459" top="759" right="482"
bottom="787">2</char>
                  </line>
            </text>
            <text id="5" left="625" top="757" right="915" bottom="791">
                  <value>Nf6</value>
                  <line left="692" top="757" right="769" bottom="790">
                        <char left="692" top="760" right="716"
bottom="790">N</char>
                        <char left="722" top="757" right="741"
bottom="790">f</char>
                        <char left="745" top="759" right="769"
bottom="790">6</char>
                  </line>
```

Figure A - 3 The received XML format from the ABBYY API after "processFields" is called with the previous Input XML format.

## 9.5 Test scenarios

This chapter contains all the test scenarios for the different subchapters inside of the Implementation chapter. No result is discussed, only what was defined and what was obtained because of this definition.

### 9.5.1 Removing of leading numbers

The test setup for the in chapter 5.3.3, the removal of an enumeration is described below.

**First test settings:**

Table A - 13

Parameters for the first ABBYY test.

| Parameter | Description |
|---|---|
| Function | process_fields |
| Letterset | RNBQKOabcdefghx12345678=+#- |
| Regular expression | (((\|([RNBQK](\|[a-h])(\|[1-8])(\|x)\|[a-h](\|[1-8])x))[a-h]([1-8]\|[18]=[RNBQ]))\|O-O(\|-O))(\|[+#]) |

**Output**

Table A - 14

Output of the first ABBYY test.

| Row | Without numbers | |
|---|---|---|
| 1 | a4 | b5 |
| 2 | axb5 | Nc6 |
| 3 | b6 | a6 |
| 4 | b7 | Ra7 |
| 5 | b8=Q | Bb7 |
| 6 | f4 | e5 |
| 7 | fxe5 | d6 |
| 8 | Qc8 | Nxe5 |
| 9 | e4 | d5 |
| 10 | Qh5 | h6 |

| With numbers | |
|---|---|
| 1a4 | b5 |
| Raxb5 | Nc6 |
| 3b6 | a6 |
| 4b7 | Ra7 |
| 5b8=Q | Bb7 |
| Bf4 | e5 |
| 7fxe5 | d6 |
| 8Qc8 | Nxe5 |
| ge4 | d5 |
| 10ah5 | h6 |

| | | |
|---|---|---|
| 11 | Bc4 | dxc4 |
| 12 | Nc3 | Ng6 |
| 13 | d3 | Bd6 |
| 14 | Nf3 | Nf6 |
| 15 | O-O | O-O |
| 16 | Qxd8 | Kh8 |
| 17 | Nd1 | Bxe4 |
| 18 | Bd2 | Nd5 |
| 19 | Qxg6 | Re8 |
| 20 | Qxe8+ | Bf8 |
| 21 | Qxf8# | |

| | |
|---|---|
| K8Bc4 | dxc4 |
| 12Nc03 | Ng6 |
| 13d3 | Bd6 |
| 14Nf3 | Nf6 |
| 150-0[23] | O-O |
| 16Qxd8 | Kh8 |
| 17Nd1 | Bxe4 |
| 18Bd2 | Nd5 |
| 1gQxg6 | Re8 |
| 20Qxe8+ | Bf8 |
| 21Qxf8# | |

**Second test settings:**

Table A - 15

Parameters for the second ABBYY test.

| Parameter | Description |
|---|---|
| Function | process_fields |
| Letterset | RNBQKOabcdefghx0123456789=+#- |
| Regular expression | (\|[1-9](\|[0-9])\\s)(((\|([RNBQK](\|[a-h])(\|[1-8])(\|x)\|[a-h](\|[1-8])x))[a-h]([1-8]\|\|[18]=[RNBQ]))\|O-O(\|-O))(\|[+#]) |

**Output**

Table A - 16

Output of the second ABBYY test.

| Row | Without numbers | |
|---|---|---|
| 1 | a4 | b5 |
| 2 | axb5 | Nc6 |
| 3 | b6 | a6 |
| 4 | b7 | Ra7 |
| 5 | b8=Q | Bb7 |

| With numbers | |
|---|---|
| 1a4 | b5 |
| Raxb5 | Nc6 |
| 3b6 | a6 |
| 4b7 | Ra7 |
| 5b8=Q | Bb7 |

[23] Inserted 0-0 instead of O-O, not fully sure why because the alphabet allows no "zero", only capital o.

| | | |
|---|---|---|
| 6 | f4 | e5 |
| 7 | fxe5 | d6 |
| 8 | Qc8 | Nxe5 |
| 9 | e4 | d5 |
| 10 | Qh5 | h6 |
| 11 | Bc4 | dxc4 |
| 12 | Nc3 | Ng6 |
| 13 | d3 | Bd6 |
| 14 | Nf3 | Nf6 |
| 15 | O-O | O-O |
| 16 | Qxd8 | Kh8 |
| 17 | Nd1 | Bxe4 |
| 18 | Bd2 | Nd5 |
| 19 | Qxg6 | Re8 |
| 20 | Qxe8+ | Bf8 |
| 21 | Qxf8# | |

| | |
|---|---|
| Bf4 | e5 |
| 7fxe5 | d6 |
| 8Qc8 | Nxe5 |
| ge4 | d5 |
| 10ah5 | h6 |
| K8bc4 | dxc4 |
| 12Nc3 | Ng6 |
| 13d3 | Bd6 |
| 14Nf3 | Nf6 |
| 150-0[24] | O-O |
| 16Qxd8 | Kh8 |
| 17Nd1 | Bxe4 |
| 18Bd2 | Nd5 |
| 19Qxg6 | Re8 |
| 20Qxe8+ | Bf8 |
| 21Qxf8# | |

### 9.5.2 ABBYY common error evaluation

This chapter describes the ABBYY common error evaluation with a self-gathered data set of 20 scorecards. This evaluation took place, before the letterset and regular expression for ABBYY was altered. The alternation removed the "0" and the "/" out of the regular expression and letterset. Following the test scenario is described. A total of 20 scorecards were filled out by 16 different writers. The total list of tested characters can be found below.

**Scorecard composition:**

Table A - 17

Composition of the scorecards inside the test set.

| Quantity | Type |
|---|---|
| 12 | Without extensions, only a full game of chess. |
| 8 | With extensions of rarely used characters (K, R, h, g, 1, 7) |

[24] Still inserts 0-0 as zeros and not capital o.

**Total combined character list:**

Table A - 18

Combined characters of the data set composition.

| Character | Quantity |
|---|---|
| R | 72 |
| N | 160 |
| B | 120 |
| Q | 140 |
| K | 52 |
| O | 80 |
| a | 96 |
| b | 120 |
| c | 100 |
| d | 180 |
| e | 140 |
| f | 120 |
| g | 64 |
| h | 76 |
| x | 180 |

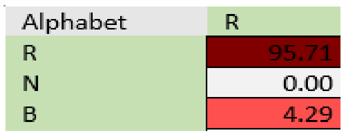
| Character | Quantity |
|---|---|
| 0 | 0 |
| 1 | 44 |
| 2 | 36 |
| 3 | 76 |
| 4 | 120 |
| 5 | 160 |
| 6 | 180 |
| 7 | 68 |
| 8 | 160 |
| = | 20 |
| + | 20 |
| # | 20 |
| - | 40 |
| / | 0 |

**Evaluation:**

To evaluate the transformations (common errors/mistakes) made by ABBYY, a Excel document is referenced. The document contains a matrix with x and y being the alphabet. All **horizontal** characters of the alphabet are the **true** characters on the scorecard.

It can be read **column by column** like this:

For all recognized characters "R" by ABBYY, 95.71% were a true character "R" and 4.29% were a character "B" which would be a wrong recognition. This can be seen in the table below.



| Alphabet | R |
|---|---|
| R | 95.71 |
| N | 0.00 |
| B | 4.29 |

Figure A - 4 ABBYY evaluation table example.

**Info:** The following tables show the transformation matrix for the whole alphabet, recorded on the data set of 21 scorecards with ABBYY's recognition. The columns show that i.e. 95.71% of all recognized "R" were real "R" on the scorecard and 4.29% of those were a misclassified "B" on the scorecard.

Table A - 19

Part one of the ABBYY character transformation.

| Alphabet | R | N | B | Q | K | O | a | b | c | d | e | f | g | h | x |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | 95.71 | 0.00 | 0.00 | 0.00 | 8.20 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| N | 0.00 | 100.00 | 0.00 | 0.00 | 1.64 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| B | 4.29 | 0.00 | 99.04 | 0.00 | 0.00 | 0.00 | 0.79 | 9.16 | 0.87 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| Q | 0.00 | 0.00 | 0.96 | 99.21 | 0.00 | 0.00 | 9.52 | 0.00 | 0.87 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| K | 0.00 | 0.00 | 0.00 | 0.00 | 85.25 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| O | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 100.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 4.55 | 0.00 | 0.00 |
| a | 0.00 | 0.00 | 0.00 | 0.79 | 0.00 | 0.00 | 73.02 | 0.00 | 0.87 | 0.00 | 0.00 | 0.00 | 4.55 | 0.00 | 0.00 |
| b | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 87.79 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 6.25 | 0.00 |
| c | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 86.09 | 0.57 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| d | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 2.38 | 0.00 | 0.00 | 99.43 | 0.00 | 0.00 | 0.00 | 2.50 | 0.56 |
| e | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 6.96 | 0.00 | 99.24 | 0.00 | 2.27 | 0.00 | 0.00 |
| f | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 1.74 | 0.00 | 0.00 | 99.15 | 2.27 | 0.00 | 0.00 |
| g | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 13.49 | 1.53 | 2.61 | 0.00 | 0.76 | 0.85 | 86.36 | 0.00 | 0.00 |
| h | 0.00 | 0.00 | 0.00 | 0.00 | 3.28 | 0.00 | 0.00 | 0.76 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 91.25 | 0.00 |
| x | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 99.44 |
| 0 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 1 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 2 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 3 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 4 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.79 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 5 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 6 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.76 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 7 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 8 | 0.00 | 0.00 | 0.00 | 0.00 | 1.64 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| = | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| + | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| # | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| - | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |

Table A - 20

Part two of the ABBYY character transformation

| Alphabet | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | = | + | # | - |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| N | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| B | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| Q | 25.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| K | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| O | 50.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| a | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| b | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| c | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| d | 0.00 | 0.00 | 2.44 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| e | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| f | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.56 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| g | 0.00 | 0.00 | 2.44 | 1.27 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| h | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| x | 0.00 | 0.00 | 4.88 | 0.00 | 0.00 | 0.00 | 0.00 | 1.43 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 0 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 1 | 0.00 | 93.48 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 5.26 | 0.00 | 0.00 |
| 2 | 0.00 | 0.00 | 87.80 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 3 | 0.00 | 0.00 | 0.00 | 94.94 | 0.00 | 0.00 | 0.00 | 1.43 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 4 | 0.00 | 0.00 | 2.44 | 0.00 | 100.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 5 | 0.00 | 0.00 | 0.00 | 2.53 | 0.00 | 100.00 | 0.00 | 0.00 | 0.63 | 0.00 | 0.00 | 0.00 | 0.00 |
| 6 | 0.00 | 4.35 | 0.00 | 0.00 | 0.00 | 0.00 | 99.44 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 7 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 97.14 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 8 | 0.00 | 0.00 | 0.00 | 1.27 | 0.00 | 0.00 | 0.00 | 0.00 | 98.75 | 0.00 | 0.00 | 0.00 | 0.00 |
| = | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 100.00 | 0.00 | 0.00 | 2.50 |
| + | 0.00 | 2.17 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.63 | 0.00 | 94.74 | 0.00 | 0.00 |
| # | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 100.00 | 0.00 |
| - | 25.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 97.50 |

### 9.5.3 Evaluation of threshold definition

Screenshots were taken from the certainty distribution in different board states. Legal moves are compared to the move candidates and the certainty value indicates how sure the algorithm is, that this legal move is the best possible guess for the available move candidates.

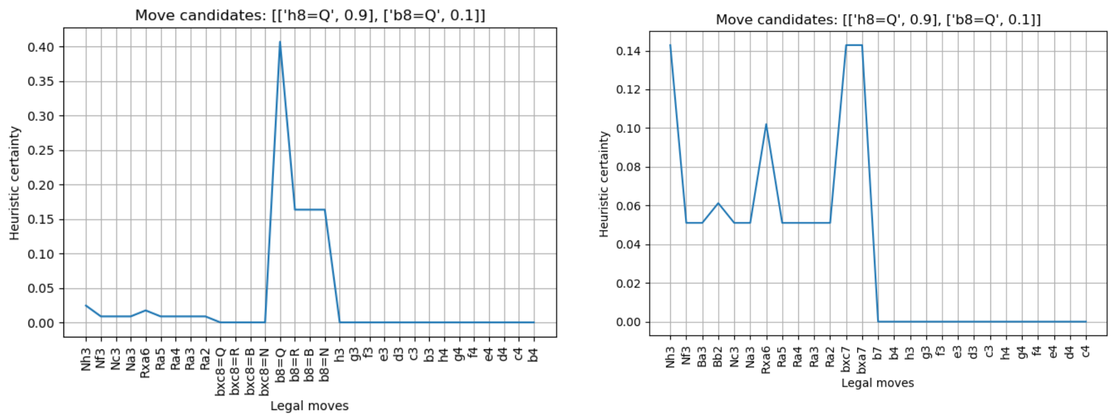


Figure A - 5 Different heuristic certainty distribution with move candidates “h8=Q” and “b8=Q”, based on the legal moves of different board states.

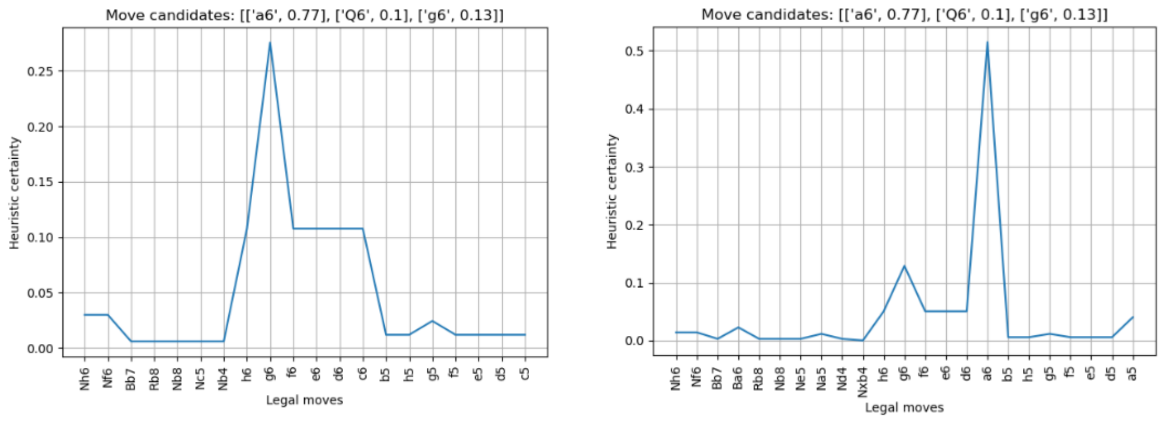


Figure A - 6 Different heuristic certainty distribution with move candidates “a6”, “Q6” and “g6”, based on the legal moves of different board states.